NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTING VOICE RECOGNITION AND NATURAL LANGUAGE PROCESSING IN THE NPSNET NETWORKED VIRTUAL ENVIRONMENT

by

Edward Michael DeVilliers

September 1996

Thesis Co-Advisors: Nelson D. Ludlow, Ph.D
John S. Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1996	Master's Thesis

4. TITLE AND SUBTITLE
Implementing Voice Recognition and Natural Language Processing in the NPSNET Networked Virtual Environment

5. FUNDING NUMBERS

6. AUTHOR(S)
DeVillliers, Edward Michael

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Interfaces to military Virtual Reality (VR) systems, such as NPSNET IV.9, have been limited mainly to keyboard, mouse, and joystick devices. This presents two major problems; remembering how to access all the functionality of the system, and using the interface when the user is otherwise physically constrained. This can occur during the use of body-position tracking devices and Heads-Up-Displays (HUD). Voice recognition and Natural Language Processing (NLP) were used as a solution to both problems.

The approach taken was to develop a networked Spoken Language System (SLS) using a Commercial-Off-The-Shelf (COTS) voice recognition and NLP system. The Nuance Speech Recognition System from Nuance Communications was chosen after analyzing the special requirements of NPSNET. Implementing the SLS occured in four phases. First, vocabularies and grammars were developed to simulate the 108 keyboard commands, focusing on flexibility and decreased response latency. Second, new C++ classes were written to ease reuse of the Nuance API's. Third, a control panel was written to manage the voice processing, and fourth, the code was integrated into NPSNET.

As a result of this effort, a new voice-enabled interface exists for NPSNET. In addition, C++ classes exist to ease future use of the Nuance API in other software systems. All of the 108 keyboard commands are executable through voice control with a 83.8% sentence understanding rate in a noisy background environment.

14. SUBJECT TERMS	15. NUMBER OF PAGES
NPSNET, DIS, 3D, visual simulation, network, distributed, virtual world, Voice Recognition, Natural Language Processing, NLP, Nuance	194
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

IMPLEMENTING VOICE RECOGNITION AND NATURAL LANGUAGE PROCESSING IN THE NPSNET NETWORKED VIRTUAL ENVIRONMENT

Edward M. DeVilliers
Captain, United States Marine Corps
B.S. Chem., United States Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

Authors: _____
Edward M. DeVilliers

Approved by: _____
Nelson D. Ludlow, Thesis Co-Advisor

_____
John S. Falby, Thesis Co-Advisor

_____
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Interfaces to military Virtual Reality (VR) systems, such as NPSNET IV.9, have been limited mainly to keyboard, mouse, and joystick devices. This presents two major problems; remembering how to access all the functionality of the system, and using the interface when the user is otherwise physically constrained. This can occur during the use of body-position tracking devices and Heads-Up-Displays (HUD). Voice recognition and Natural Language Processing (NLP) were used as a solution to both problems.

The approach taken was to develop a networked Spoken Language System (SLS) using a Commercial-Off-The-Shelf (COTS) voice recognition and NLP system. The Nuance Speech Recognition System from Nuance Communications was chosen after analyzing the special requirements of NPSNET. Implementing the SLS occured in four phases. First, vocabularies and grammars were developed to simulate the 108 keyboard commands, focusing on flexibility and decreased response latency. Second, new C++ classes were written to ease reuse of the Nuance API's. Third, a control panel was written to manage the voice processing, and fourth, the code was integrated into NPSNET.

As a result of this effort, a new voice-enabled interface exists for NPSNET. In addition, C++ classes exist to ease future use of the Nuance API in other software systems. All of the 108 keyboard commands are executable through voice control with a 83.8% sentence understanding rate in a noisy background environment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENT

I would like to thank the NPSNET Research Group for providing the funds to acquire the Nuance Voice Recognition System which made this thesis technically possible.

My most heart-felt thanks goes to my wife, Cathy Medina, and my children, Sara and Eric, whose inspiration, love, and sacrifice, not to mention the many hot pitchers of coffee, kept me going. Thanks also to my parents, whose support for me and my family during trying times really made a difference.

I would also like to thank my thesis advisors, Major Nelson Ludlow, USAF and John Falby for their guidance and attention to detail which contributed to the overall worth of this work. Thanks also go out to John Locke, Randall Barker and Paul Barham, who gave me their time and knowledge in NPSNET.

And thanks be to God, from whom all good things come.

I. INTRODUCTION

In the field of Virtual Environments (VE), one of the major areas of research has been the Human-Computer Interface (HCI). While much effort has been placed in the design of haptic, locomotion, and full-body interfaces, and the use of position tracking, there has been relatively little work done in the use of voice or speech recognition in these environments. Although voice recognition is not a priority in current VE research [NRC96], its use can have a tremendous impact in the use of Virtual Environments through the use abstraction, the reference of unseen entities, and the ability to use hands and eyes for other purposes. The focus of this thesis is to implement a Spoken Language Understanding (SLU) system for a virtual environment, specifically the NPSNET Networked Virtual Environment being developed at the Naval Postgraduate School (NPS) [ZYDA94]. This thesis will examine the special requirements of VE's, the strengths and limitations of voice recognition systems in this environment, and finally presents an implementation architecture of a networked SLU system for NPSNET using the Nuance voice recognition system.

A. SPOKEN LANGUAGE UNDERSTANDING

1. What is Spoken Language Understanding

Spoken Language Understanding is the melding of two closely related areas: voice or speech recognition and natural language processing. Voice recognition deals with the conversion of voice input into a stream of words that can be further used by an application. This usually encompasses using hardware and Digital Signal Processing (DSP) techniques to convert voice into the best matching words in a predefined vocabulary. Natural Language Processing (NLP) uses known data about the words and predefined rules that show authorized word structures to give the meaning of what was said. Together, they allow people to take advantage of the flexibility of natural, albeit limited, language to accomplish tasks.

2. Importance of Spoken Language Understanding Systems

One of the most natural methods people have of interacting within the world is through the use of spoken language. Another is through the use of body movement and the direct manipulation of objects. Contrasting the two methods can show the usefulness of Spoken Language interfaces.

Haptic interfaces use body movement, especially that of the hands, to interact with objects in the virtual world, or objects in the real world through the use of telerobots. Examples include the cyberglove and the joystick. Haptic interfaces are gaining increased attention. A National Research Council report has recommended increased funding and research in this area [NRC96] since they provide a greater immersive effect for the user into the virtual world. However, haptic interfaces have several limitations.

- First, they only interact with objects nearby in the virtual environment.
- Second, the performance of haptic interfaces is dependent on the abilities of the user. For instance, the coordination of the user is important in conveying the exact actions needed.
- Third, haptic interfaces deal with concrete objects. They are not useful for abstract concepts, such as time, quantities, relative positions, and descriptions.
- Finally, while in use, it can be hard to use hands for other purposes, such as issuing keyboard queries and commands.

The use of spoken commands can counteract each limitation mentioned above. In addition, spoken language can be used in conjuction with haptic interfaces to form multimodal interfaces that combine the best of both interface paradigms while adding new capabilities to the system.

B. MOTIVATION

NPSNET currently does not have a built-in speech interface ability. However, it is a prime candidate for a voice interface because of its use as a testbed for different VE applications and interfaces. NPSNET is currently being used as a base for work in networked vehicle simulation [ZYDA94], inserting a Virtual Soldier in the battlefield [ZYDB95] [WALD95], simulating naval damage control [OBYR95], and as a Navy

Officer of the Deck (OOD) trainer [NOBL95]. Each application could benefit from the use of a voice interface. For instance, inserting a soldier into the virtual battlefield would require the soldier to be hooked up to a motion tracking device, such as Polhemus or Ascension Bird motion trackers, or a SARCOS uniport or treadmill device. In addition, he would normally be carrying a weapon, such as an M-16 rifle. Obviously, he would not be able to input, by himself, one of over a hundred keyboard commands available to affect the simulation. Even if he could, the number of keyboard commands is too large to be effectively remembered, especially by a new user or someone in an emergency situation. Currently, very limited voice recognition has been added to the OOD trainer, showing the utility of voice recognition in this environment [STEW96].

NPSNET, as a military simulator, provides other challenges to the use of voice recognition. It includes spatial sounds [STOR95] that provide battlefield effects via external speakers. This makes the inclusion of voice recognition systems with their weakness for background noise an important research issue. Additionally, while SLU systems typically deal with two dimensional (2-D) stand-alone environments, NPSNET is a three dimensional (3-D) networked virtual environments. Therefore, NPSNET is an outstanding platform to use to test the utility of SLU systems to such VE's.

C. GOALS AND APPROACH

The main goal of this thesis is to implement a spoken language understanding system within NPSNET. Additionally, the architecture of this system is designed to work in a networked environment, to be easily maintained through its life cycle to handle new network protocols, and to be less impacted by changes in the underlying NPSNET system. This thesis does not expand the ability of current voice recognition products, but examines the feasibility and efficiency of, and resolves problems concerning, the incorporation of an off-the-shelf SLU system in a networked, 3-D, virtual environment.

The approach taken to meet the stated goals was:

- Review the current state of voice recognition technology. In conjunction with an analysis of the NPSNET environment and its uses, determine what kind of

Current Off-the-Shelf (COTS) voice recognition system is most appropriate. This resulted in choosing the NUANCE voice recognition system from SRI International. It is a client-server, continuous speech, speaker independent voice recognition system which has already been used in LeatherNet, a current research project being sponsored by the Office of Naval Research and Development (NRaD)[BRAT96].

- Examine the current input commands available for NPSNET. This effort focused mainly on the keyboard commands implemented within the system.
- Develop a set of spoken commands to initiate NPSNET keyboard commands and system queries. For the initial vocabulary, the NPSNET User Guide was examined for examples of the type of vocabulary a new user might use to control the system.
- Come up with a grammar which can be used with the above vocabulary to give a more flexible and natural command language.
- Analyze the current architecture of NPSNET IV and the proposed architecture for NPSNET V to determine a suitable architecture for the SLU system.
- Implement the SLU system in C++ using the above developed architecture.
- Disambiguate and generate appropriate commands based on the input speech string.

D. THESIS ORGANIZATION

This thesis is organized into the following chapters:

- Chapter I: Introduction. This chapter gives a general outline of the work, including the major objective, the motivation behind the thesis, the approach taken which will be expanded upon in proceeding chapters, and the organization of the thesis.
- Chapter II: Background and Previous Work. Current and past systems that relate to the research conducted in this thesis are discussed. This includes an introduction to NPSNET, Voice Recognition concepts and functional groupings, and previous work on integrating voice recognition and NLP with human computer interfaces.
- Chapter III: Implementation Questions using Voice Recognition in NPSNET. This chapter discusses the main ideas and questions raised in this thesis. It examines interrelated factors that require compromise in the use of voice recognition in virtual environments. It then examines important questions concerning the production of grammars for virtual environments, and what factors inherent in virtual environments need to be considered.
- Chapter IV: NUANCE Voice Recognition System and System Requirements. This chapter presents an overview of the architecture, capabilities, and system requirements of the NUANCE voice recognition system.

- Chapter V: The SLU System Architecture. This chapter explains the design of the SLU system that will interface with NPSNET. It discusses the developed grammar, design decisions and their possible impact.
- Chapter VI: Grammar Development. This chapter examines the natural language rules developed to implement the SLU system. It looks at how the representation problem was handled, and what factors were varied to improve the response time of the SLU System.
- Chapter VII: Conclusion. The results and lessons learned in this thesis are discussed, as well as future work.

II. BACKGROUND AND PREVIOUS WORK

Since the days that the television (TV) series Star Trek first went on the air, people have dreamed of having the same power and flexibility in the use of computers as the crew did on the Starship Enterprise. Instead of being tied to a keyboard, the computer could hear you and understand your commands. In a way, that TV series has set the standard in how we view the advent and progress of voice recognition technology and Natural Language Processing (NLP). This chapter discusses the field of voice recognition, especially its current technological state. It then examines the NPSNET Virtual Environment (VE) which will be used to explore the use of spoken language systems in VE's. Finally, it reviews examples of voice recognition technology used today in virtual environments.

A. VOICE RECOGNITION

1. Overview

Voice recognition, sometimes referred to as speech recognition, is a general term for the use of voice input in several different types of applications. In this thesis, it is used to denote the processing of voice input and producing the words spoken. In this context, it becomes only the bottom level of a larger spoken language understanding system (Figure 1). Voice recognition systems use computer hardware to convert voice input into digital data which can then be analyzed using Digital Signal Processing (DSP). The DSP output is used to choose the best word match among an application's vocabulary. This is called lexical analysis as seen in Figure 1. Once words are chosen, data about the words held in a lexicon is used to perform syntactic and semantic analysis (levels 3, 4, and 5 in Figure 1) to find the meaning of what was said. Higher level functions then attempt to understand the meaning of voice input from prior sentences (i.e., reference, anaphora, ellipsis, and speaker-listener models) and using known contexts. These different processes can occur in parallel to one another.

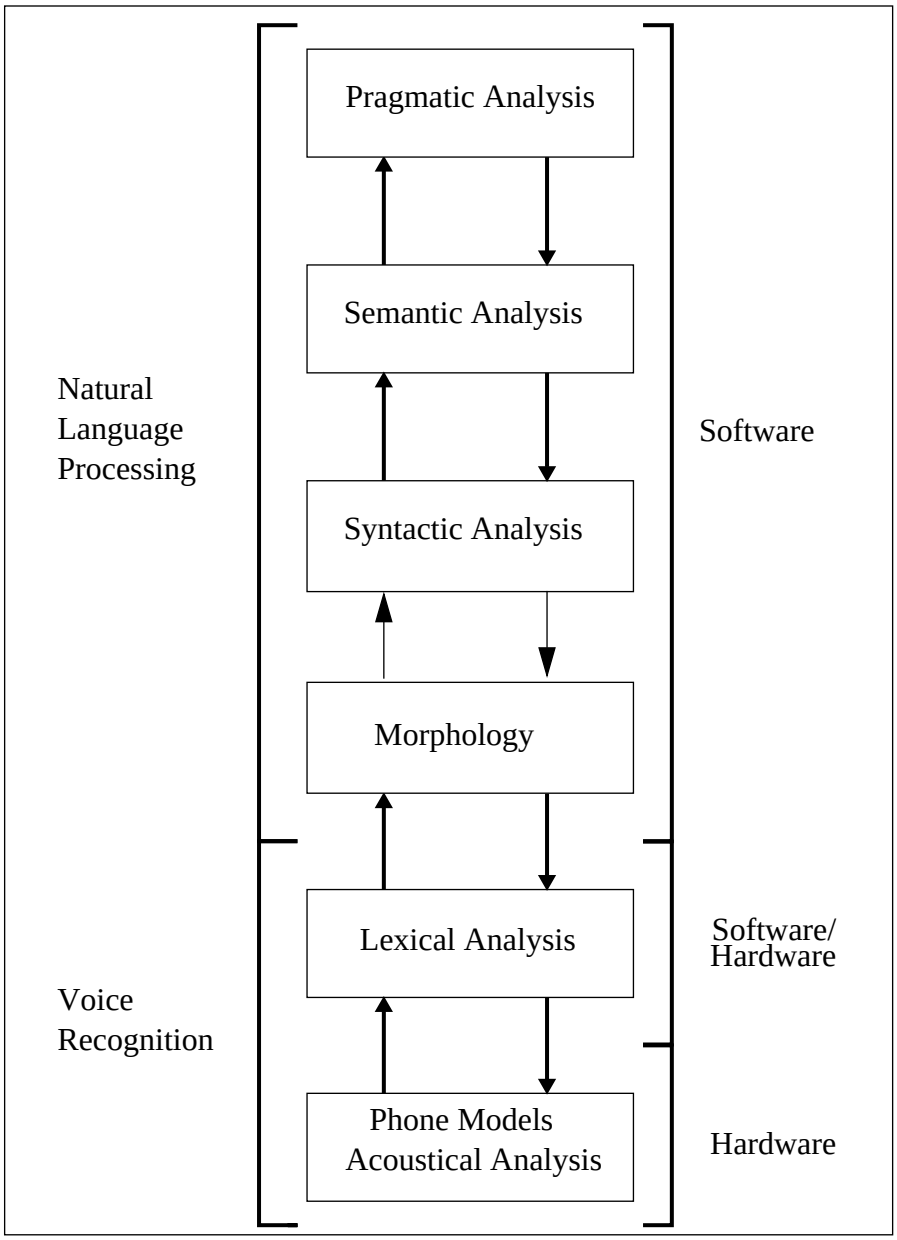

Figure 1. Levels in Spoken Language Understanding Systems

2. Voice Signal Characteristics

The human voice is both the focus and the main obstacle of automated voice recognition. Unlike humans, the computer does not have other senses, such as sight, that it

can use to make out speech. For instance, it cannot lip read to help determine what was said, or see who just spoke. All the information must be extracted from the speech signal alone. This input can be extremely variable from speaker to speaker, even when saying the same words. This is because of the methods used to articulate speech, the types of sounds made while speaking, and the ways used to process the acoustic signal.

Human articulation uses the mouth, throat, and nose, and the structures in these parts of the body, to produce distinguishable sounds called phonemes (Table 1). The throat and vocal cords produce the voiced phonemes such as “ee”. The mouth and nose are resonating cavities that affect the overall speech signal produced by adding a rich blend of frequencies. In addition, the mouth has several points of articulation (the teeth, alveolar ridge, and the hard and soft palate) and articulators (the lips and tongue) that help produce the various phonemes we hear. Since the structure of the mouth, nose, throat, teeth, etc. varies from human to human, the voice signal itself will vary between speakers. This is self evident since we can distinguish one person’s voice from another.

Articulation produces voice signals that are complex waves made of both cyclic (sinusoidal) and acyclic waves composed of different frequencies. Each phoneme produces a different combination of frequency ranges and cyclic/acyclic waves. The resonating cavities also produce secondary harmonic frequencies called formants whose relatively constant position and separation from each other give important clues to a phoneme’s identity. These characteristics can be used to classify phonemes as seen in ARPABET [MARK96], and identify them in the acoustic signal. However, a phoneme is affected by other phonemes, silence, and stops that come before and after it since the vocal tract cannot instantaneously move from one position to another. This forms diphthongs which are the combination of two phonemes placed together.

3. Digital Signal Processing

The acoustic signal produced by human articulation produces complex waveforms that hold large amounts of information. This information includes the main frequency of

Vowels		Consonants		
Phonemes	**Example**	**Type**	**Phoneme**	**Example**
ah	father	Frictive (voiced)	v	very
ae	tap		th	there
aw	talk		z	zebra
a	bay		zh	beige
eh	step	Frictive (unvoiced)	f	fast
uh	run		th	thing
ee	beep		s	seek
i	lift		sh	show
oh	tone		h	hit
oo	moon	Plosives (voiced)	g	get
oo	book		d	dither
er	stir		b	base
		Plosives (unvoiced)	k	cat
			t	two
			p	poke
		Nasals	n	no
			m	me
			ng	ring
		Glides	y	you
			w	will
		Semivowels	l	last
			r	real

Table 1: American English Phonemes

the signal, the rate of change of the signal frequency, the amplitude or loudness of the signal, and the formants. It also includes noise that affects all the waves, and frequencies

that span a wide range. Therefore, the processing of voice signals must sample a subset of the frequency range that will give the most relevant information, while at the same time try to suppress the noise that would interfere in the signal analysis.

In the analog to digital conversion of a voice signal, the analog signal must be sampled for frequency and amplitude. The rate of sampling is usually twice the highest target frequency because of Nyquist's theory. This way, the beginning, middle and end of the highest frequency wave is captured. For voice processing, the range is usually around the 100 Hz to 4000 Hz, producing approximately a 8000 Hz sampling rate.

The samples must be further compressed, since 8000 samples are still too much information. Therefore, the samples are grouped into small, constant time blocks called frames. The samples inside the frame are then analyzed to produce discrete values that can be used to deduce the phonemes or words that were spoken in the original voice signal. This analysis can be done in two major ways. First, Fast Fourier Transforms (FFT) can be applied to the signal which is initially divided into frequency bands. The FFT defines each frequency band in terms of its component frequency waves. Second, Linear Predictive Coding (LPC) or one of its variations can be used, such as Cepstral Coefficients or Vector Quantization. These methods use prior frame data to produce sets of coefficients or vectors that can be used to determine the spoken phoneme. These methods are currently the dominant means of coding the spectral data, and are less demanding of computational power and storage. [MARK96] Once the voice input has been coded into a finite set of coefficients, the system is then ready to start the recognition process.

4. Hidden Markov Models (HMM)

Hidden Markov Models are currently the most prevalent technique used to recognize spoken input. Developed independently at Carnegie Mellon University [BAKE75] and IBM [MARK96], it has superceded prior recognition methods such as template matching and acoustic-phonetic modeling. HMM's are a network of finite state machines made up of states and transitions with associated probabilities. The transitions are

governed by two sets of random variables - one, X, is the state of the model at time t, and the other, O, is the output symbol at time t. Which state the system is in at any given time is hidden.

Each state has two probabilities, b and a, the probability of generating a specific symbol ‘k’ given a particular state, and the probability of transitioning from one state, ‘i’, at time ‘t’ to another state,’j’, at time ‘t+1’.

$$b_i(k) = p(O_t = k \mid X_t = i) \qquad \text{(Equation 1)}$$

$$a_{ij} = p(X_{t+1} = j \mid X_t = i) \qquad \text{(Equation 2)}$$

HMM’s use the coefficients produced from FFT’s or LPC’s as input into the network. The recognition system can then use the observed input to go through the HMM’s to see which one produces the best match, i.e., which HMM models the input the closest and is the best choice. The system can also search through the network to see which path through the network matches the observed input the closest. This best path approach is called the Viterbi Algorithm.

Whether we try to find the best match or best path, we must first find the probabilities to be used within the HMM. An iterative algorithm exists called the Baum-Welch Algorithm which can be used to train the system. Initial values for probabilities ‘a’ and ‘b’ are used, and then, using reestimation formulas, new values are produced. Reestimation is performed again, and the process continues until the values converge within a predefined value. The Baum-Welch Algorithm is proven to produce better results the more it is used. [BAUM72]

Training is the key to HMM’s. Since the observed input is determined by the signal analysis, the HMM system needs to be trained on a system resembling the target platform. For example, a speaker independent system should be trained using a variety of speakers, both male and female. Current work shows that this could require as little as three or four people [VEEC95]. Speaker-independent systems can also include several speaker models,

one for males, one for females, and possibly others for geographical pronunciation differences. However, the environment is also important. Background noise needs to be modeled so it can be ignored. Equipment, especially the specific microphone used, can also affect the probability values produced during training. The implication to voice recognition in virtual environments is that current HMM systems may need specialized training to increase accuracy.

5. Categories of Voice Recognition Systems

Voice recognition systems can be categorized by two major factors: training requirements and speech flow. These two categories form four general areas. A voice recognition package can either be speaker independent or speaker dependent. A speaker dependent system needs to be trained to a specific person's speech patterns and characteristics. The voice system can then either accept continuous speech, or it must work with discrete speech where the speaker needs to insert a small pause in between each word. However, there are other important factors which come into play which an interface designer must take into account. These include the perplexity, i.e. the size of the vocabulary a speech system can choose from at any time, the acceptable error rate, and the ratio of processing time to speech rate (Figure 2). These factors must be considered in order to create a workable application.

For this thesis, there were certain goals set dealing with the different factors mentioned above. As a natural language system, continuous speech with some casual word inclusion was seen as a necessity. This means that users had some leeway on the choice and use of articles, adjectives, and some descriptive phrases. As a control interface to a military simulation, decreasing processing time was also important. This meant trying to cut the perplexity of the grammar to about 10 words, and accepting a higher error rate by using faster but less accurate acoustic models, which will be discussed in Chapter IV. If faster processing rates become necessary in the future, we must accept an even higher error rate,

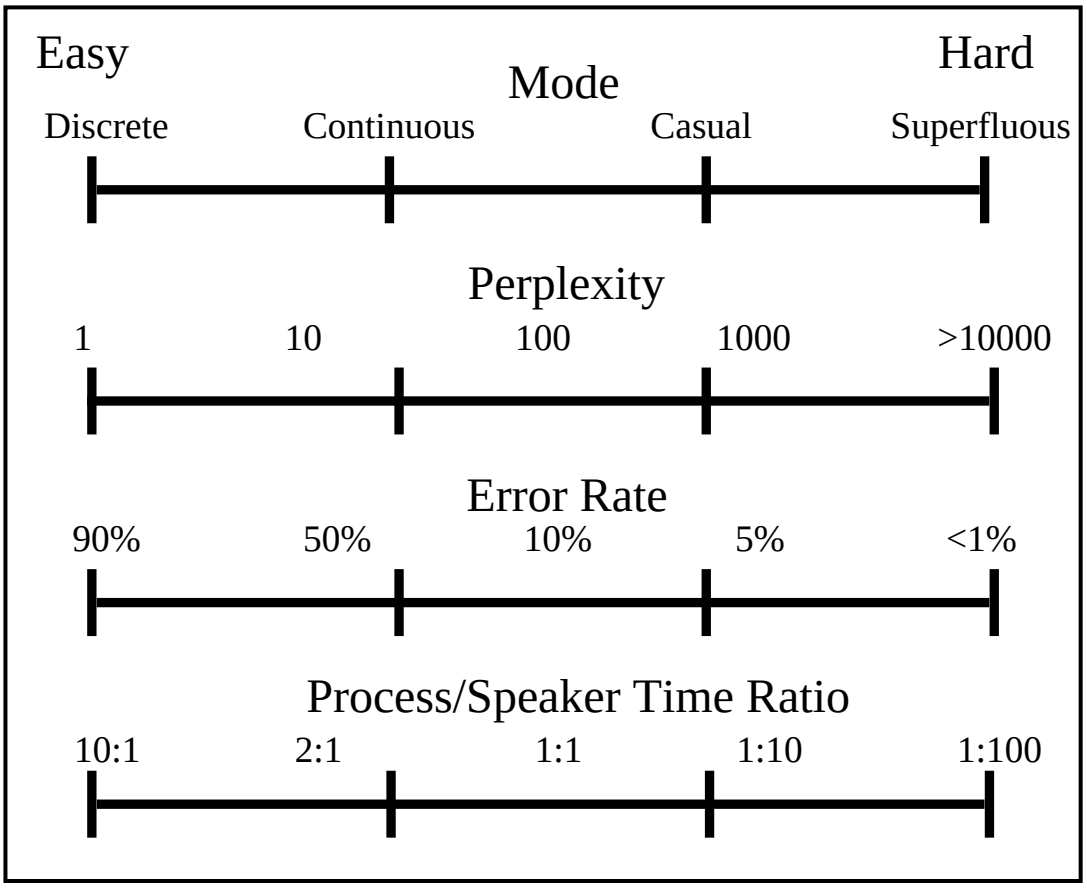

Figure 2. Voice Recognition Factors

or decrease the perplexity again. This would result in lessoning the ability to have a more casual dialogue with the voice system.

6. Current Voice Recognition Applications

Voice recognition has been used in the following application areas:

- Dictation
- Command and Control
- Data Entry
- Data Access
- Telephony

Each one of these areas have different requirements in terms of speaker independence, continuous versus discrete speech flow, vocabulary size, speaker training,

and environmental conditions, such as background noise. This thesis deals with command and control of the virtual environment interface. Command and control systems usually require speaker independence with a quick response time and very low error rate [MARK96].

B. NPSNET

1. Overview

NPSNET is a large scale, distributed, virtual environment developed at the Naval Postgraduate School [MACE94]. It is used as a testbed for implementing the Distributed Interactive Protocol (DIS) for DoD simulations, and for incorporating new input devices important in virtual environments. These devices include flight control sticks (FCS), Head Mounted Displays (HMD), Polhemus Position tracking, and the SARCOS I-Port. NPSNET is implemented in C/C++ and in the Performer Graphics API developed by Silicon Graphics, Inc. [SGIA95].

The NPSNET-IV architecture has separate buffers for both the network packets coming from other applications, and for input signals originating from the host machine (Figure 3). Since the DIS protocol is limited in the type and amount of information it can

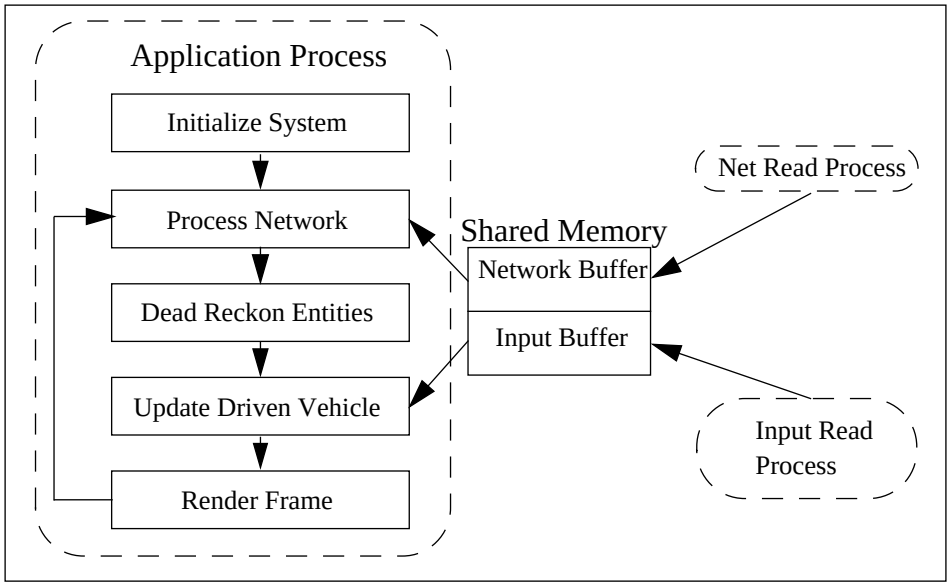

Figure 3. NPSNET-IV Architecture

pass to distributed simulations, other network protocols have been developed that also have their own buffers and network managers. This includes the Information Data Unit (IDU) protocol and the High Resolution (HIRES) protocol [BARK96]. These protocols are used extensively in applications developed at the Naval Postgraduate School which are based on NPSNET IV, such as the Submarine Trainer [BACO95] and the Shipboard Walkthrough [STEW96]. The challenge in implementing a networked voice input front end to NPSNET is to find ways to use these different network protocols and input buffers to easily implement the commands described in detail in Chapter IV.

2. Interface Design

NPSNET-IV is designed to present DIS entities, such as tanks, and static objects such as roads, buildings and terrain, as three dimensional objects. The interface also allows two dimensional information, such as position, heading, radar information, and weapon status, to be printed on the screen (Figure 4). This is similar to the Heads-Up Display (HUD) in fighter aircraft. The keyboard input device allows users to toggle the display of this information, along with the presentation of graphical effects such as fog and anti-aliasing. [CSD96]

3. Current Uses of NPSNET

NPSNET is used by over a hundred commercial and government institutions. Additionally, it has been used as the basis of several training and battlefield visualization applications. These include a control measure visualization using a virtual sand table [KIRB95], a submarine trainer with distributed, multi-user controls [BACO95], a physically-based helicopter trainer [LENT95], and a shipboard damage control and OOD trainer [STEW96].

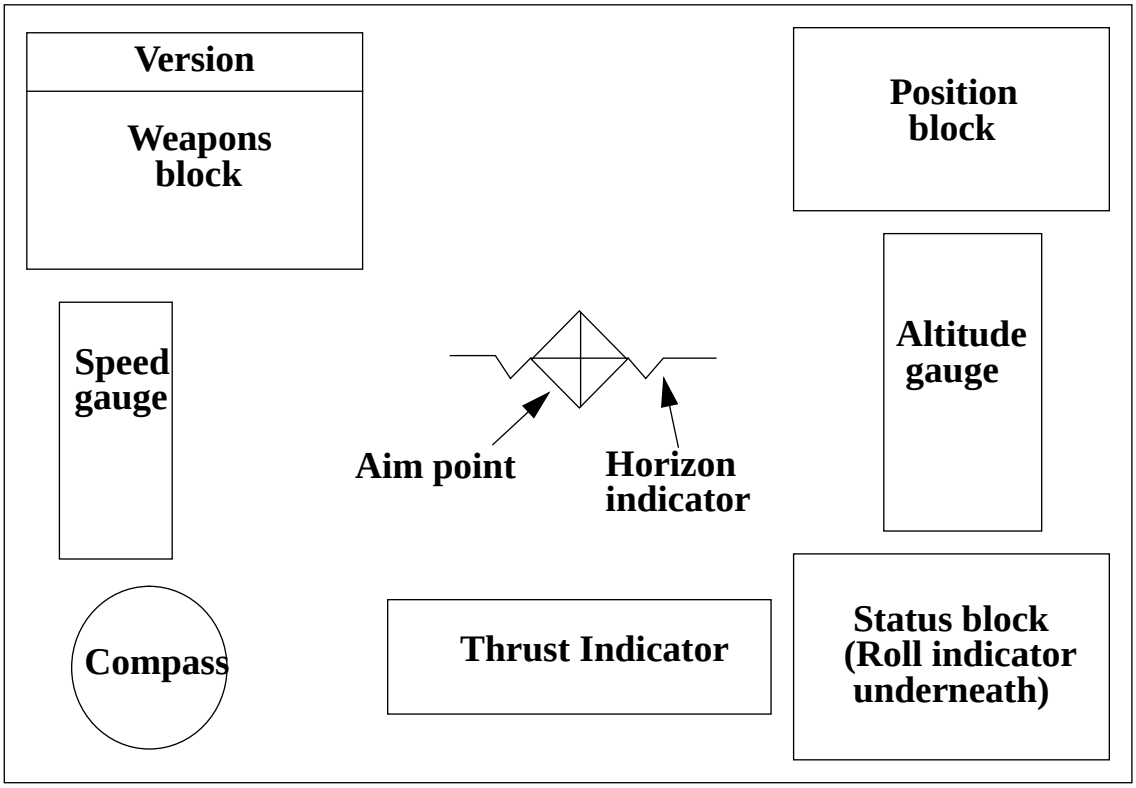

Figure 4. NPSNET Screen Information Layout. From [CSD96]

C. PREVIOUS WORK

1. Desktop Control and Dictation

Two of the most popular applications of voice recognition technology among IBM compatible machines and other architectures are desktop control and dictation. Companies such as Dragon Systems, Creative Labs, and Verbex Voice Systems have marketed voice systems that use common sound hardware to control the PC desktop (Table 2). These systems closely mimic the menu metaphor by providing a graphical metalanguage. For instance, if a menu choice says “OPEN”, the voice command to choose that menu option is “open”. Little, if any, natural language processing is used. The majority of these systems are discrete speech and speaker dependent. Dictation systems, such as Dragon Dictate and IBM’s Voice Type Dictation, are exclusively speaker dependent, discrete-word systems.

Company	Title	Type	URL
Dragon Systems	Dragon Dictate	Spkr Dep/ Discrete	www.dragonsys.com
Kurweil Applied Intelligence	Kurweil for Windows	Spkr Dep/ Discrete	www.kurz-ai.com
Verbex Voice Systems	Listen for Windows	Spkr Dep/ Discrete	www.txdirect.net/ verbex
Command Corp	In Cube	Spkr Indep/ Discrete	www.commandcorp.com
Speech Systems, Inc.	Phonetic Engine	Spkr Indep/ Continuous	www.speechsys.com
PureSpeech	PureSpeech Recognition Engine	Spkr Indep/ Continuous	www.speech.com
Articulate Systems	PowerSecretary	Spkr Dep/ Discrete	www.artsys.com
Voice Processing Corporation	VPro/Continuous VPro/PRL VProXD	Indep/Cont Indep/Both Indep/Disc	www.vpro.com
Apple	PlainTalk	Spkr Indep/ Discrete	ftp.support.apple.com/
Silicon Graphics, Inc.	SppechManager	Spkr Dep/ Discrete	www.sgi.com
BBN	Hark	Spkr Indep/ Continuous	www.bbn.com/bbn_hark/ HarkHome.html
Nuance Communications	Nuance	Spkr Indep/ Continuous	www.sri.com

Table 2: Partial List of Voice Recognition Vendors and Systems

2. Hearsay II

The Hearsay-II system is a product of the Defense Advanced Research Projects Agency - Speech Understanding Research project. Developed at Carnegie Mellon

University in the mid 1970's, Hearsay-II was a speaker-dependent, hardware-augmented speech understanding system [ERMA80]. It used a blackboard model where a central module or process coordinated the use of knowledge bases found in the system. The information in these knowledge bases, such as phonetics and phrase level structures, are used to create hypotheses about segments of the voice input. The knowledge bases coordinate with each other via the blackboard.

3. NAUTILUS and Eucalyptus

The Naval Research Laboratory (NRL) has conducted several projects examining NLP used in conjunction with current GUI systems and virtual environments. Such projects include the InterFIS project [EVEB92], the NAUTILUS project [WAUC96], and the Eucalyptus and InterVR projects [EVEA96]. Eucalyptus combines speech recognition into a GUI interface while InterVR combines speech recognition into a virtual reality system. These projects focus on the issues of spoken language understanding and use commercial voice recognition systems to provide the initial voice input that the systems then process. Hence, they parallel this thesis's approach of examining the use of Spoken Language Understanding in current GUI/VR systems.

a. NAUTILUS

A central component to NRL's research efforts in NLP is NAUTILUS, the Navy AUTomated Intelligent Language Understanding System. It performs the natural language processing required in NLP projects. It has several characteristics which affect the performance and range of abilities of the Eucalyptus and InterVR systems based on this component.

- It is modular and sequential. NAUTILUS is built using the following modules: PROTEUS, the syntactic parser developed by New York University [GRIS86], the TINSEL semantic interpreter developed at NRL [WAUA90], FOCAL for reference/anaphora resolution, and FUNTRAN (FUNctional TRANslator) which converts the TINSEL and FOCAL output into procedural calls. In processing the voice input, each module executes in the order outlined above. An error at any point will cause the processing to fail.

- It is syntax driven. Since PROTEUS performs the initial processing, input which is not well-formed according to the given grammar will cause the parse to fail. Although PROTEUS is a chart parser, the results of partial parsing contained in such a parser cannot currently be retrieved.
- Output is based on a regularization language. Although there are many ways to say the same thing, PROTEUS and TINSEL output and/or input data in a regularized form based on lambda conversion [ALLE95].
- It uses procedural semantics. Each predicate which takes an argument, such as an adjective or verb, maps directly into a Lisp function.

Several lessons can be learned from NAUTILUS. The modular design makes it simple to improve one area of the natural language processing. However, each model is constrained by the expectations and output of the individual modules. The sequential use of these modules makes it very difficult, and currently impossible with this system, to use the partial parse results to retrieve any information. While applying NAUTILUS to different applications, the researchers found that the grammars could be divided into two major groups, a sub-grammar that was common across different domains, and another sub-grammar that is application specific [SAGE86]. This is similar to the results found by other researchers in voice driven interfaces. [MOOR89]

b. Assumptions and Design of Eucalyptus

The Eucalyptus system was started to expand on earlier NL research efforts to interface to the KOALAS Airborne Early Warning (AEW) Test Planning Tool, a proof of concept application written by the Los Alamos National Laboratory. Eucalyptus was designed with the following principles in mind:

- Voice Interface mirrors the GUI. The voice interface does not add any functionality to the original GUI. It is assumed that the GUI contains all the functionality needed.
- Input must be natural language dialogue. The voice interface only accepts conversable natural language.
- Discourse tracking for input only. The system only tracks the user's input, not the system's output. Any references to output results will not be understood.
- Restricted Domain. The grammar is designed to handle references, commands, and queries of a very limited domain, namely the AEW domain.
- Works on the set-theoretic and logical operations (denotative reference) and

context maintenance (discourse tracking).

- Modular Design. Interface change only needs to change one part. (Figure 5)
- Multimodal. Uses voice, GUI, and Deictic (gesture) input.
- No Graphical Metalanguage. Since the GUI and voice interfaces are parallel and equivalent, the voice interface is designed not to mimic GUI actions, such as “Open Experimenter Control Panel, and Push Aircraft Trails Switch” [WAUB94]. Eucalyptus does not let the user interact directly with GUI controls via voice.
- Supports imperative and query statements.

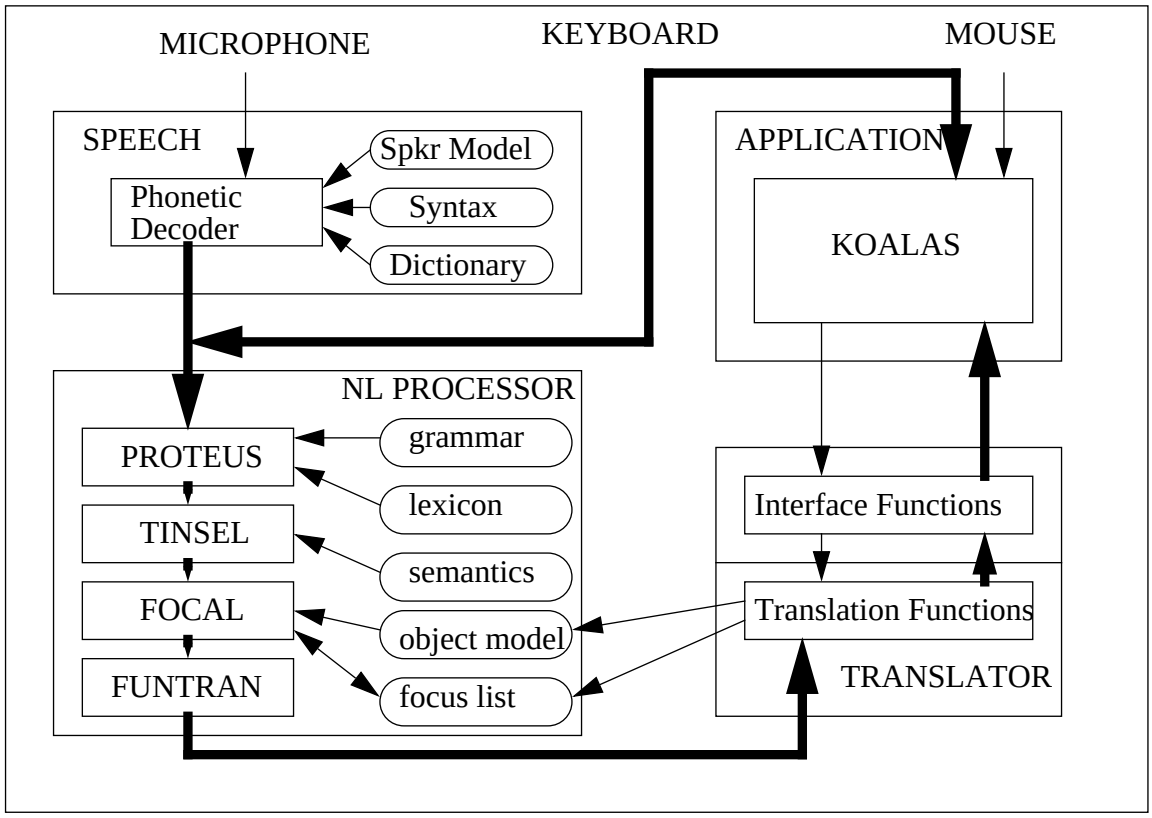

Figure 5. Eucalyptus Architecture [WAUB94]

c. Lessons Learned

Several lessons were learned from adding a voice interface into an existing GUI system. These are:

- GUI menu systems map well into verb phrases. The menu structure paralleled voice commands or imperatives, such as “SHOW”, “AIRCRAFT TRAILS.”

Structures such as check boxes were each assigned a verb.

- Ellipsis and anaphora increased naturalness and ease of use. However, the use of ellipsis and anaphora had to be defined in the grammar, and were not the result of partial parsing.
- Voice input and the GUI were complementary. Users switched back from one to another continually.
- Users did not always want to converse. Users wanted to use a graphical metalanguage to refer to specific menu items as is common in PC speech recognition systems.

4. LeatherNet and CommandTalk

LeatherNet is a combat simulation system being developed for the United States Marine Corps by Naval Research and Development (NRaD) and SRI International. It is composed of three major parts: CommandVu which is based on NPSNET, modified Modular Semi-Autonomous Forces (ModSAF) software from Loral Corporation, and CommandTalk which is a NL interface to ModSAF (Figure 6). The purpose of CommandTalk is to allow relatively untrained Marines to interact with the LeatherNet system using natural language in order to view battle plans and tactics. In this way, it augments the training already accomplished at the Combined Arms Staff Trainer at the Marine Corps Air-Ground Combat Center during workups for Combined Arms Exercises (CAX). However, it is envisioned that LeatherNet will provide more flexibility and insight into tactical decisions and strategy by the use of a three dimensional virtual environment.

Overall, LeatherNet is a distributed environment that uses three different protocols to talk to its different parts. The Distributed Interactive Simulation (DIS) protocol is used to communicate between CommandVu (NPSNET) and ModSAF so that CommandVu can display the entities in the VE. Currently, the CommandTalk voice interface does not interact with CommandVu directly. Therefore two operators are needed, one for CommandVu and the other for ModSAF/CommandTalk. The Persistent Object (PO) protocol is a ModSAF protocol that was originally created for different ModSAF stations to coordinate with each other the actions and data of entities they control. In order to communicate with ModSAF, CommandTalk also uses the PO protocol. Finally, the Open

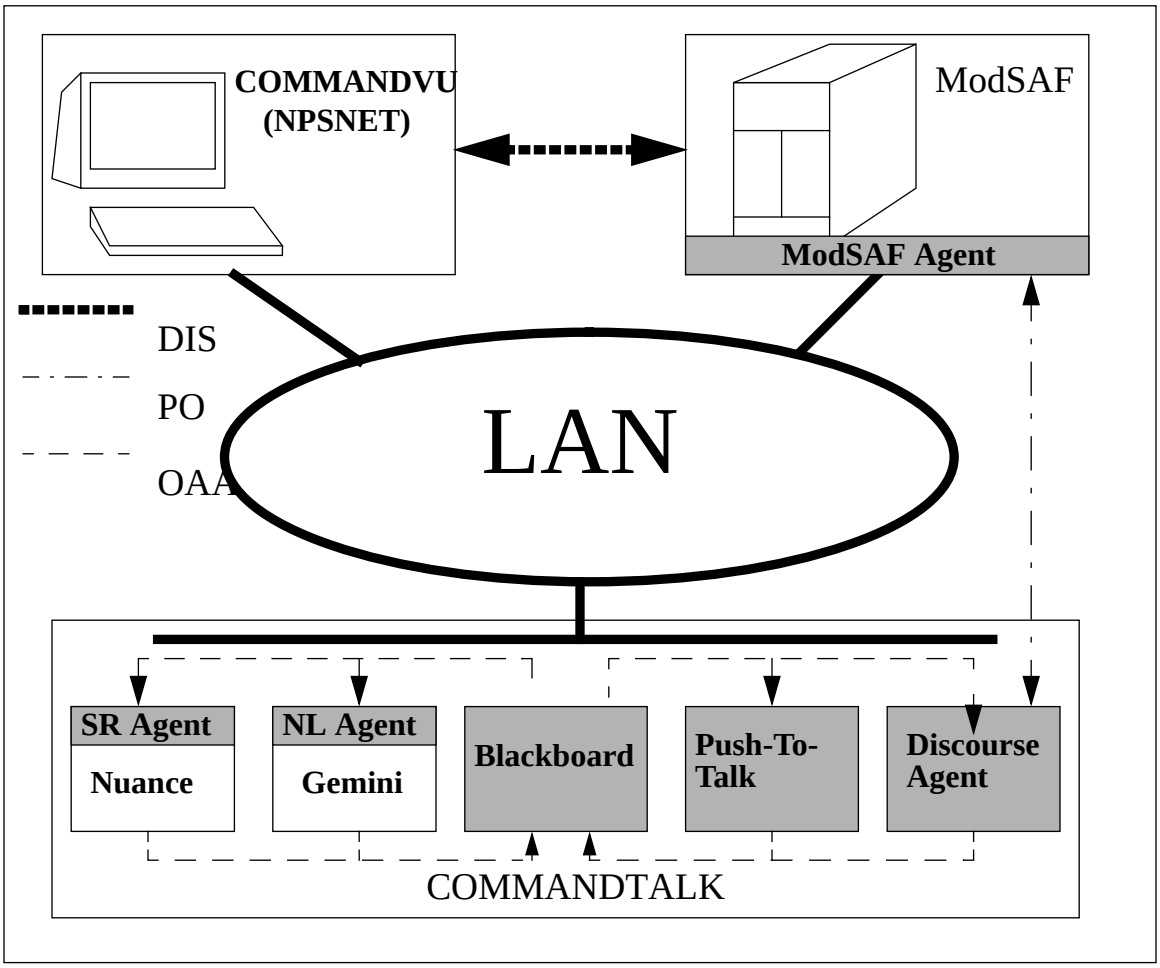

Figure 6. LeatherNet and CommandTalk Architecture

Agent Architecture (OAA) protocol is used internally between the individual CommandTalk components (agents).

CommandTalk is an agent-based voice interface system that is used to give commands to ModSAF in order to create, control, and delete entities in the virtual environment. The major agents are the Speech Recognition (SR) agent, the Natural Language (NL) agent, the Push-To-Talk agent, ModSAF agent, the Discourse agent, and the blackboard. Like the Hearsay-II system, CommandTalk uses a blackboard approach to coordinate among the different agents since the agents work asynchronously with each other. For example, a new voice command can be processed by the SR and NL agents while

the Discourse and ModSAF agents are still processing the last command. The blackboard makes the NL agent wait to send its result to the Discourse Agent until the Discourse Agent notifies the blackboard that it is ready.

CommandTalk holds several similarities to the Eucalyptus system mentioned previously. They are both modular, reduce input into a logical form representation, and interact with a GUI. The Discourse Agent, along with the ModSAF Agent, does different types of resolution, such as predicate resolution. This is similar to the procedural semantics requirements of Eucalyptus, where the verb “move” may mean different things in different contexts.

However, there are several significant differences. First, CommandTalk is not complementary to the ModSAF GUI. CommandTalk is meant to replace the GUI in order to increase ease of use for relatively untrained users. This is accomplished through the use of a ModSAF agent which augments the functionality of the original ModSAF GUI. The underlying code was modified so the agent could make calls to routines that normally would not be accessible from the GUI. The system developed in this thesis made as few changes to NPSNET as possible. Only the functionality inherent in the current NPSNET system is available.

Secondly, the SR and NL agents in CommandTalk cooperate together to form the logical form. However, Eucalyptus does redundant work. In Eucalyptus, the Speech Engine is separate from the syntactic parsing and semantic analysis. However, the Speech Engine, like most speech recognition systems, use a grammar to reduce the perplexity of the word search during the recognition process. Therefore, syntactic processing occurs twice, once within the Speech Manager and a second time within the Proteus system. The Nuance voice recognition system used in this thesis combines the syntactic and semantic analysis much like the CommandTalk system. This way we can reduce the processing time for executing voice commands.

III. IMPLEMENTATION QUESTIONS USING VOICE RECOGNITION IN NPSNET

The methodology of using voice recognition in user interfaces is still being developed. This thesis explores how voice recognition and natural language processing can be used to more effectively interface into a networked, three-dimensional virtual environment. This involves addressing certain questions. These include surveying the available voice recognition technology, and determining which, if any, Current Off-The-Shelf (COTS) packages may be useful. Once a COTS package has been chosen, we can turn toward examining what tasks in the VE can be controlled using voice technology, and what grammars can be developed to execute these tasks. In order to make this interface usable by the greatest number of people, both trained and untrained in using the system, we also need to look into natural language processing to see how it can be used to give the user more flexibility when interacting with the system. Both the grammars and natural language processing must be tailored toward flexibility and quick response time, which lead to compromises between these two factors.

A. CHOOSING APPROPRIATE VOICE TECHNOLOGY

The NPSNET Virtual Environment (VE) is a distributed simulation with 108 keyboard commands to control the system's actions and display. Being distributed, it can be used by any number of different speakers who may be located both locally and across the country. Therefore, the software should be speaker independent. Usually, speaker-independent systems have demonstrated lower accuracy levels than speaker-dependent systems which conduct user training. Speaker-dependent systems have an easier task of analyzing one voice pattern, rather than trying to use several other factors from the signal analysis to take into account the variations found in different human voices. However, many speaker-dependent systems are used in dictation systems, where the decreased computational load using speaker dependence balances the problem of large perplexity.

Command and control systems have commonly used discrete speech or voice macros (phrases that correspond to user commands) to accomplish tasks [MARK96]. Accuracy and quick response time are paramount in these systems. Hence, many command and control systems use discrete speech. Some systems, such as SGI's SpeechManager, try to hide their discrete nature by using voice macros that allow phrases to represent desktop actions. However, these phrases are treated essentially as very large words. The system cannot parse the phrase into its component words and then analyze them for meaning or use.

While discrete systems may be more accurate, they suffer from several problems, as do the keyboard and menu control voice systems they are supposed to replace. First, memorization of commands becomes a large problem as the functionality of an application increases. Currently, NPSNET has over 100 keyboard commands using one, two and three key combinations. Obviously, this results in a large learning curve for new users, and the need for frequent refresher training. Menu systems, although easing the user's memory requirements, still places memory demands on the user in order to find the needed menu choice. Discrete voice systems that only mimic the keyboard with short one or two word commands still place a large memory requirement on the user. They now need to remember over a hundred word commands. Discrete systems that use the vocabulary found on menu choices still retain the same problems that menus incur.

Natural language processing systems solve many of these problems found in discrete systems used in command and control. These types of systems may give the user the added flexibility needed to handle the large functionality of NPSNET. However, the use of natural language brings other problems which must be discussed. One large problem is that the user does not know what range of language is acceptable to the NL system. This implies that the development of useful grammars becomes even more critical.

Another problem is how the method of speech input (discrete or continuous speech) affects the user of VE systems. There is no current empirical data on the effects of discrete speech versus continuous speech input. Discrete speech systems require some restraint by

the user in order for there to be clear word separation boundaries. This possibly could cause problems for the user in terms of concentrating on the VE and maintaining the suspension of disbelief. In order to study this question, we would have to make two similar SLU systems, one using discrete speech and the other using continuous speech, which is beyond the scope of this thesis. In this study, we decided that the question concerning the effects of discrete versus continuous speech input would detract from the overall effort to create a SLU system that could work in a VE. Therefore, this thesis focused on the use of speaker-independent, continuous-speech voice systems. We then studied how well a current COTS system handled command and control functions in NPSNET.

B. DEVELOPING GRAMMARS

The development of grammars is critical to the efficient use of a voice interface. A badly formed grammar may result in mostly unrecognized or misinterpreted commands. The main question becomes how do we form an efficient grammar? One way is to observe and record the interaction users have with the system to be modeled. For example, a military command and control system that mimics the manual operations found in a Combat Operations Center (COC) may have its vocabulary based on what dialog is recorded during normal user interactions with each other and the current system. The disadvantage is the time required to gather the utterances into a corpus, and to perform the analysis of the corpus for sentence structure and vocabulary.

In this thesis, we started by using a corpus already made up of sample user commands. This corpus is contained in the user's guide [CSD96]. As experience is gained in the actual use of the voice recognition system, the grammar can be iteratively improved. As the grammar is developed, we need to examine the following areas: grammar ambiguity, confusable words, and grammar efficiency (depth versus breadth).

C. USE OF NATURAL LANGUAGE PROCESSING

The ability to conduct NLP is dependent on the inherent functionality of the COTS system used, in this case Nuance version 4.0. The thesis examined what NLP ability is

found in Nuance, and how it may be used to handle the following; syntactic/semantic processing, meaning representation and a common logical form, and disambiguation. The ultimate goal of any NLP system is to extract an unambiguous meaning from the given input.

The first NLP question addressed in this thesis was the use of semantic grammars. As will be covered in Chapter IV, Nuance uses a user-defined grammar to help not only in word determination, but in meaning recognition. The grammar used can be defined in many ways: syntactically or semantically. The type of grammar used affects the way we can extract the input's meaning. This thesis examines the use of semantic grammars since these grammars lead to less ambiguity during meaning interpretation and work well in applications with specific domains of knowledge [ALLE96].

The thesis explored different ways that the case-frame approach to meaning representation could be used in Nuance. Built-in Nuance features, such as slot definitions, templates, and classes, were used to develop a case-frame system.

D. HANDLING NOISE AND SOUND EFFECTS

Work has been done in NPSNET to add spatial 3-D sound [STOR95]. Spatial sound has several advantages in VE's, such as increasing the immersive effect, and providing information such as the location of entities that are not within the user's field of view [NRC96]. However, it provides additional challenges to the use of a voice interface.

Work was done to see if the use of spatial sound and other sound effects had a serious negative effect on voice recognition accuracy and speed. This included trying different HMM acoustic models (discussed in Chapter IV and VI) that may incorporate different background noise levels, using different types of microphones, and reviewing the grammar to check for confusable words.

E. GENERATING NPS COMMANDS FROM THE SPEECH SYSTEM

Generating legal NPS commands and executing them is a three part problem. The first problem, which is related to an NLP problem already mentioned, is how we get a

common representation of different voice commands that aim for the same affect? For example, the commands “Turn on the fog” and “Fog on” aim for the same affect, but syntactically they are very different. We may want the final representation to be “Command: Change State; State Name: Fog; State Value: On.” Once we have such a representation, we can use a case-based routine to send the proper command. The second problem then becomes how the command can be sent to NPSNET. Thirdly, once the command is received by NPSNET, it becomes a large implementation problem of finding where the needed routines are.

F. LIMITATIONS TO THE THESIS

There are several areas which were beyond the scope of the thesis which may easily be included as follow-on work. These areas are:

- This thesis does not seek to improve on current voice recognition systems. Such work could include using toolkits to create custom HMM’s to accurately represent background noise, typical stress levels, etc. Instead, COTS systems will be used exclusively to provide the basic voice recognition needed.
- The SLU system produced in this thesis is targeted for command and control of the VE only. We focused on imperative commands, and not on querying the system.
- The use of anaphora, i.e., pronouns and other references to past discourse, is not supported. While grammatically the use of anaphora would probably not be difficult, the supporting framework to keep track of past discourse and disambiguate references would take too long to develop.
- The Speaker-Listener model we use is constrained to the military VE control domain. This limits the needed grammar and makes disambiguation of the spoken commands easier.
- Deictic commands, i.e., the use of gestures to explain the meaning of words such as “here” or “there” is not supported. The joining of gesture and voice input is an important area of interface research [COHE94], but it is not needed to implement the desired commands, which are most of the keyboard commands.

IV. NUANCE SPEECH RECOGNITION SYSTEM

The Nuance Speech Recognition System from Nuance Communications, Inc. was chosen as the COTS tool to implement the SLU system of this thesis. It is a speaker-independent, continuous-speech, client-server based, C-language API framework with a built-in dictionary of 20,000 words. It has three speech models to choose from for increased recognition accuracy. It can be used to create both procedural and event-driven applications.

A. ARCHITECTURE

The architecture of Nuance can be divided into two parts; the grammar package generation process (Figure 7), and the recognition client/server. The grammar package

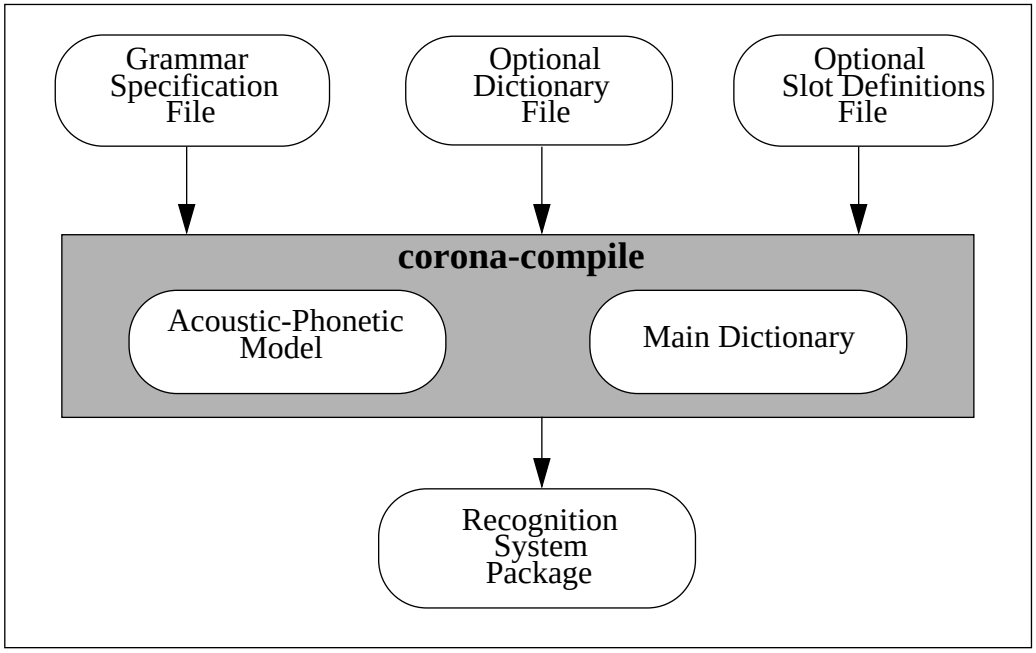

Figure 7. Grammar Package Generation Process

gives specific information about the vocabulary and grammar rewrite rules, and NLP rules and assumptions of the application. Hence, the grammar package gives the application its

"character" or uniqueness that distinguishes it from other applications. The grammar package provides the following:

- Grammar File. This contains all the non-recursive grammar rewrite rules that define what can be said by the application's users. It also holds rules showing how meaning and values can be extracted from the defined grammatical structures, and how these values should be stored (see the Slot Definition File below).
- Dictionary File. An optional file, it contains phonetically spelled words needed by the application that were not included in the standard Nuance dictionary.
- Slot Definition File. An optional file, it holds the names of variables called slots that can hold the results of Natural Language Processing as defined in the grammar file.
- Acoustic-Phonetic Model. The speech input will be analyzed in comparison to an acoustic-phonetic model that contains data using a certain frequency range, input device (such as specific microphones), and different statistical models that need varying amounts of processing power. [NUAN95]

The recognition client/server architecture is based on the following objects based on C language structures: the CoronaConfig object, the NLEngine object, the RecClient object, and the RecServer object. The general relationship between these objects and the users in general are shown in Figure 8. These objects use other structures, such as RecResult and NLResult to store information the system has produced.

B. VOICE RECOGNITION CAPABILITIES

The voice recognition abilities of the Nuance system come from the combined use of the speaker acoustic models, the application's grammar file, and the system and user defined dictionaries. Nuance uses Hidden Markov Models (HMM) as a basis for its voice models. As discussed in Chapter II, HMM's are the most popular means of low level signal processing to determine word usage. The training an HMM receives has a direct bearing as to how it will perform. Hence, Nuance comes with three acoustic models from which we could choose. Each one has a speed/performance trade-off. The three models are:

- Genome Models. Based on a continuous density Gaussian equations, these models are the most accurate, but also the slowest. This was the one used during much of the grammar development of the thesis.
- Phonetically Tied Mixture (PTM) Models. While also based on Gaussian

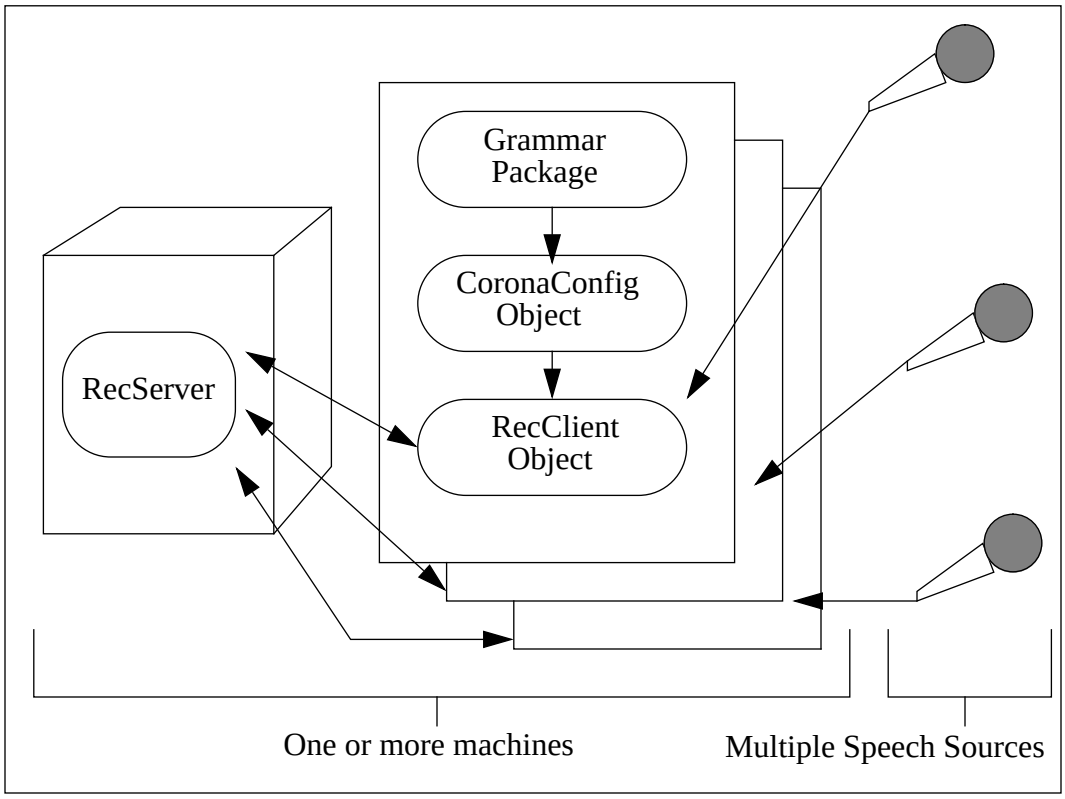

Figure 8. Recognition Client/Server

equations, the number of Gaussian coefficients is less, resulting in faster performance with decreased accuracy.

- Vector Quantized Models. These models use discrete-density probability functions. They are fastest and the most inaccurate of the three model sets.

A large problem with most voice recognition systems is the perplexity, as discussed before. In order to reduce the word search and increase performance, Nuance uses a user-defined grammar to describe what words may be possible at any point of the recognition process. The grammar is defined using Nuances's Grammar Specification Language (GSL). It is very much like the definition of a context-free grammar found in finite automata theory [ULLM95]. From a root rule, a series of rewrite rules define what may be said and when. For example, the start of the NPSNET grammar with representative rewrite rules in GSL appears in Figure 9.

```
.NPSNET-Commands
        [   Display-Options             {<category display-options>}
            Entity-Movements            {<category entity-movements>}
            Environment-Effects         {<category environment-effects>}
            Human-Movements             {<category human-movements>}
             Input                      {<category input>}
            Level-Of-Detail             {<category level-of-detail>}
            Sim-Operations              {<category sim-operations>}
            Weapons                     {<category weapons>}
        ]


Human-Movements
        [   Person-Move-Cmds            {<object soldier>
                                          <action move>}
            Rifle-Cmds                  {<object rifle>
                                          <action aim>}
            Formation-Cmds              {<object soldiers>
                                          <action signal>}
        ]


Formation-Cmds
        (   ?[form (get into)]  ?a
            [   (   echelon             {<formation echelon>}
                    [   left            {<direction left>}
                        right           {<direction right>}
                    ] )
                column                  {<formation column>}
                vee                     {<formation vee>}
                wedge                   {<formation wedge>}
                (open up)               {<formation open>}
                (close up)              {<formation close>}
            ]  ?the ?formation )
```

Figure 9. Example of a GSL Grammar

The Nuance GSL has a compact and efficient notation to describe complex grammars. The brackets'[ ]' give the logical OR choices in the grammar, while a sequential element list can be given by parenthesis '( )' that give the logical AND. Additionally, Nuance allows

its rewrite rules to contain optional elements which are marked by a preceding question mark '?'. This is especially important in creating flexible natural language grammars since definite and indefinite articles, such as 'the', 'a', and 'an', and adjectives, adjective phrases, and other parts of speech which are not necessary for understanding the speech, and may or may not occur, are common in spoken language.

These three constructs, '[ ]', '( )', and '?', can be used together in whatever order a user wishes. This results in a very small representation of complex grammars. For example, the rewrite rules for the grammar Formation-Cmds in Figure 9 would require the following rules (Figure 10) if written in Baukus-Naur Form (BNF):

```
Formation-Cmds -> form Formation
Formation-Cmds -> form a Formation
Formation-Cmds -> get into Formation
Formation-Cmds -> get into a Formation
Formation-Cmds -> Formation
Formation-Cmds -> Formation formation
Formation-Cmds -> Formation the formation
Formation -> Echelon | column | vee | wedge |
             Open | Close
Echelon -> echelon left
Echelon -> echelon right
Open -> open up
Close -> close up
```

Figure 10. Baukus-Naur Form

In GSL, the sub-grammars names are capitalized while expected words are the terminals and are written in lower case. While the GSL is very flexible, it does not allow direct nor indirect recursion of rules. Recursive rules are very common in natural language grammars. For instance,

Noun-Phrase -> Prep-Phrase

Prep-Phrase -> Preposition Noun-Phrase

While input strings can be parsed to see if they can fit a recursive grammar, the complexity of this problem can be exponential. The complexity of parsing using depth-first

search is Cn^3, where n is the number of words [ALLE95]. Therefore, the use of recursion can make the grammar inefficient in terms of time, and one of the goals of speech systems is real-time performance. However, there is research that indicates that such recursion in natural language is usually not more than three levels deep [MARC80]. Therefore, these rules can be explicitly written to capture the expressiveness of natural language.

When the grammar is completed, it can be compiled to form a package. The Nuance compiler looks up the words as they are spelled in the grammar to retrieve their phonetic spelling. Words that are not present are sent to a "*grammar-name*.missing" file, where grammar-name is the directory name is the name of the grammar file compiled by the Nuance system. The user can then give one or more phonetic spellings for each word in the grammar-name.missing file and then rename it "*grammar-name*.dictionary". The grammar can then be recompiled.

C. NATURAL LANGUAGE PROCESSING CAPABILITIES

1. Natural Language Processing (NLP)

Natural Language Processing is the processing of tokens in order to extract some meaning. Usually this corresponds to processing words that have been input by some means. This input, at first, was text based, but it is not limited to this as words can come from recognized speech input. The fundamental assumption of NLP is that the words used and their relative position to each other give information that can be to determine the meaning of the input phrase.

NLP is hard since it is a search problem with the goal of finding an unknown meaning. In practice, NLP resorts to tree structures based on the grammar rules of the language being interpreted. Since the meaning is unknown, it is difficult to know if a search through the tree structure is correct. If more than one tree can be constructed to fit the search through the input string, then the language is ambiguous, and multiple meanings can be found.

Ambiguity is the largest problem in NLP, and it takes many forms [CARB94]. The preceding example was a grammar ambiguity. Ambiguities can be categorized as follows [CARB94]:

- Word sense: “The man went to the bank to get some cash and jumped in.” The word “bank” can refer to a financial institution or the side of a river,
- Case: “He ran the mile in four minutes in the Olympics.” Case refers to the relationship between the central concept of running, with the two structurally similar prepositional phrases. The phrases refer to different properties, time and location.
- Referential: “He put the book on the table. Then he moved it.” Does ‘it’ refer to the book or the table?

There are several analysis techniques used to solve the inherent ambiguities in natural language, and extract the meaning of the phrase. There are:

- Pattern matching. This technique looks for key words, then compares the surrounding words with known patterns to interpret what has been said.
- Syntactic parsing. Using know language structures and parts of speech, such as noun and verb phrases, prepositions, and adjectives, the input phrase is fit into a parse tree that uses grammatical rewrite rules. A correct fit means that the phrase corresponds to the grammar. A parser can give the part of speech a word is, and where it was found. It still does not give explicit information about what is going on.
- Augmented Transition Networks (ATN). Additional information and rules are used to give information, such as plurality, tense, noun-verb agreement, and semantic agreement. For instance, “The dog reads the book.” could be rejected by an ATN as semantically incorrect if information about the words “dog” and “reads” included a measure of intelligence, and a rule existed to compare this intelligence measure between a subject and its verb. ATN’s quickly explode in complexity as the size of the grammar and vocabulary grow.
- Semantic grammars. Using both syntactic and semantic structures, this grammar can give information about the use of words in a phrase as the phrase is being parsed. The grammar used in this thesis is a semantic grammar.
- Case frames. Based on the work of Charles Fillmore [FILL68], case frames try to define a small, finite set of concepts (frames or slots) which can hold the meaning of a phrase. This set of frames can be related to head concepts, such as specific classes of actions, as in Fillmore’s work. It can also be used to try to define all general actions, as is done in conceptual dependency [SCHA75].

2. NLP Implementation in NUANCE 4.0

NLP in Nuance is a separate process compared to voice recognition [NUAN95]. The NL system uses a case frame type of NL processing which works in conjunction with the grammar definition of the application. Although the listing of what slots are available in an application are entered in a *grammar-name*.slot_definition file, the manner in which values are entered into the slots is defined in the same grammar file which contains the application grammar's rewrite rules. An example of this is back in Figure 9. After the voice input has been processed to give a RecResult structure, the Nuance NL engine takes the result and processes it through the same grammar. As it goes through the grammar, it encounters NL commands which are surrounded by curly braces. These commands are associated with the rule to their immediate left. If that rewrite rule is used, the NL command associated with it is executed, and the slot mentioned in that command is filled.

The Nuance system provides mechanisms to use variables to determine slot values, to restrict slot combinations with respect to values already entered, and to specify which slots are required to be filled. This is done through the use of a special {return} command, slot templates, and slot classes. In this way, a developer can have the Nuance system deal with case ambiguity.

The NL system can handle a variety of value types as slot values. These include integers, strings, and structures. Integer values can be manipulated using Nuance-provided math functions, such as "plus" and "minus". Also, integer values can be converted transparently into string representations through the use of Nuance API calls.

D. SYSTEM REQUIREMENTS OF NUANCE

The Nuance system is available for Sun Microsystems, IBM RS/6000, SGI, and PC workstations running Solaris. It also requires a minimum of 16 MB of RAM, plus that RAM needed by the application. Nuance takes up about 20 MB of hard drive space.

Since Nuance is a client-server system, it is possible to put most of the CPU load for recognition on one machine, which acts as the server, while other machines play the role of

a client that actually accepts and samples the voice input and sends it to the server. Therefore, a network connection would be required. nuance allows the port and/or IP address of the server machine to be specified. This allows the server to be anywhere on the network.

This thesis ran both the server and client processes on an SGI Indy workstation with the following characteristics:

- Irix 5.3
- One 132 MHz MIPS R4600 processor.
- 64 MB RAM
- Indy integrated DSP
- Indy integrated 24-bit graphics
- Lightspeed VR-350 headset microphone.

Further information about the Nuance Speech Recognition System can be found by contacting

Nuance Communications
333 Ravenswood Ave., Building 110
Menlo Park, CA 94025-5120
(415) 462-8200
(415) 462-8201 fax

V. SLU SYSTEM ARCHITECTURE

The Spoken Language Understanding System produced for this thesis is a network-capable application designed to meet three purposes. First, this system eases the integration of voice control by using C++ classes designed to encapsulate the details of the Nuance API, which is mainly a C-language API. Second, the system uses a control panel that provides feedback on what has been said, controls the speech environment parameters, and provides a more centralized means of developing application grammars. In short, it is a simple Integrated Development Environment (IDE). Third, the system is distributed, hence network capable. It can therefore interact with any voice capable application on the network, can interact with more than one machine at a time, and allows the processing of the voice signals on other machines. This last point is especially important in graphic-intensive environments, such as NPSNET, where the computer needs all its resources to run the graphics pipeline.

A. OVERALL VIEW

The structure of the SLU system is portrayed in Figure 11. The system was developed using the RapidApp application prototyping package for SGI systems [SGIC95]. RapidApp provided an object-oriented framework called ViewKit [SGIE94] which handled the creation of the VoiceAppDisplayClass and provided the base application class, VkApp, from which the voiceVkApp class is derived.

During the system's initialization, the main routine instantiates a voiceVkApp class object and the VoiceAppDisplayClass object, which contains the actual GUI interface and handles the X Window and GUI calls. This interface is called the Voice Control Panel and will be discussed in further detail later. Based on the command line arguments passed, voiceVkApp will instantiate the recognizerClass object which handles voice processing by using its own component objects: the configClass object, the recClientClass object, and the NLClass object. The voiceNetManagerClass object is not instantiated during system

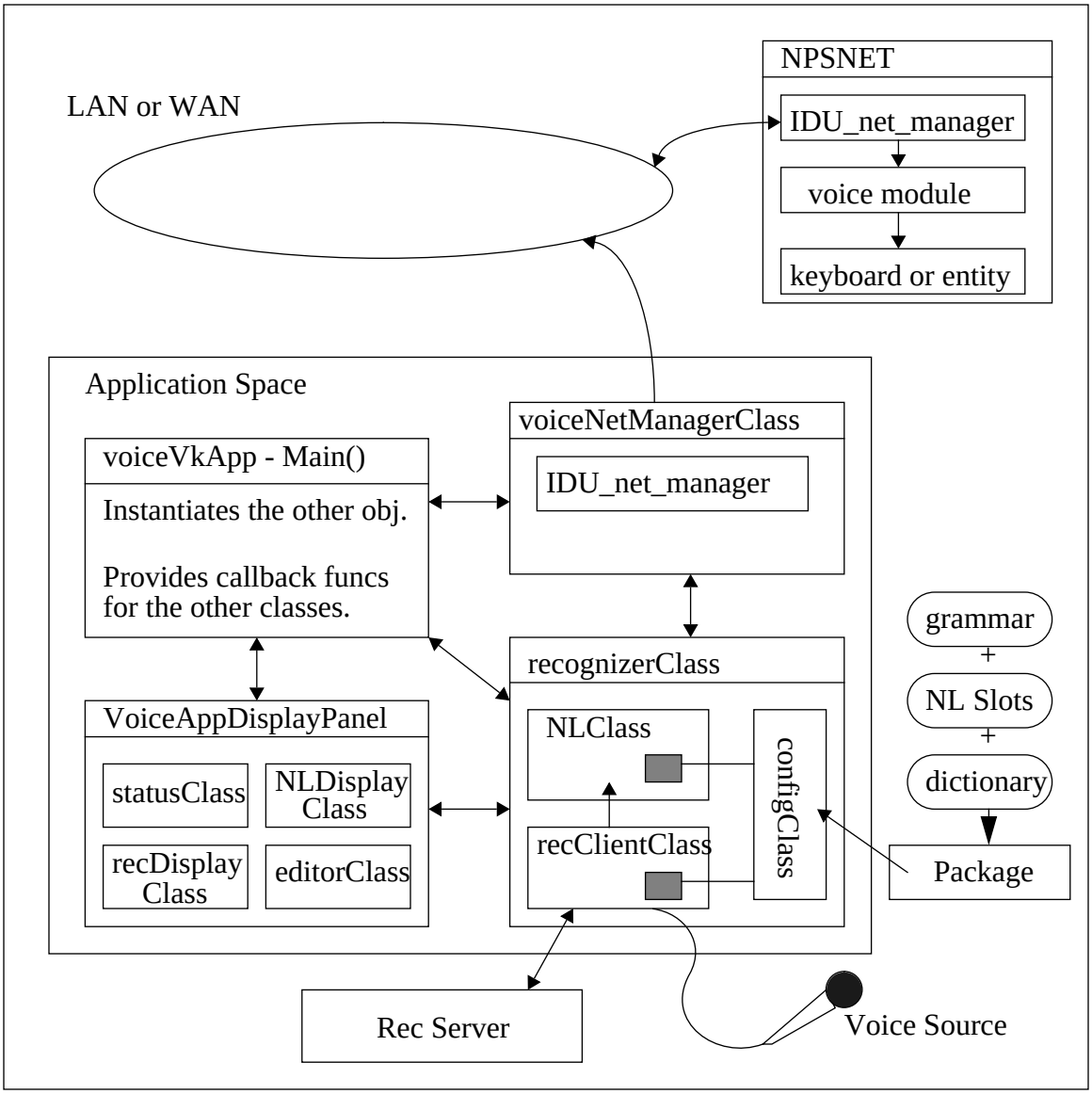

Figure 11. SLU System Diagram

initialization. It is opened from the Control Panel, and uses the multicast address passed in as a command line argument, or a default address.

The system can be started by issuing the following command: voiceApp [Nuance command line arguments] [IDU net manager arguments]. The Nuance command line arguments are covered in the Nuance Speech Recognition System Developer's Manual [NUAN95]. All Nuance arguments are available for use. The IDU net manager arguments

are listed in Figure 12. If no arguments are passed, the system only instantiates the

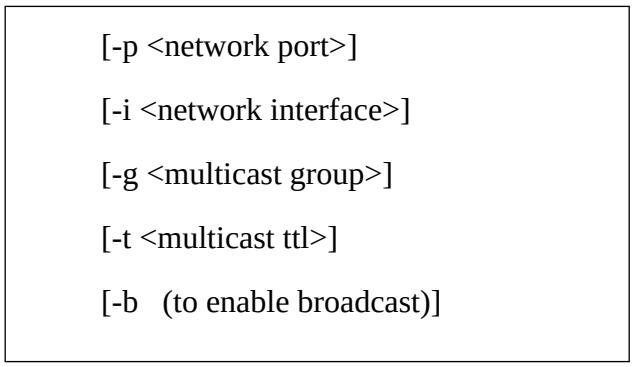

Figure 12. IDU Command Line Options

voiceVkApp and Control Panel objects. The network object will default to using multicast address 224.200.200.200 when it is instantiated from the Control Panel. The Nuance package can also be entered later via the Control Panel. Once the package directory location is entered, the system will instantiate the recognizerClass object with its underlying Nuance pieces.

VoiceApp is an event driven system. Therefore, it uses a series of callback functions to provide its functionality. The voiceVkApp object has the callbacks for the Nuance-based recognizerClass object. The VoiceAppDisplayClass object contains all the callbacks related to the GUI. These functions, in turn, can query the different objects of the system for information or to perform an action. This is made possible by the definition of four global variables in the main.C file of the application. These global variables are made available to all the objects in the system. This could not be helped, since both the X Window system and the base Nuance API's need to register callback functions. With the C language, this is not a problem. But with C++, this requires the use of static member functions which by definition cannot reference object specific data members without the object's pointer passed in [YOUN92]. The recognizerClass object and its Nuance based objects do not use these global variables since the classes were designed to be application independent.

B. OBJECT ORIENTED CLASS STRUCTURE

As stated previously, one of the goals of this thesis is to make the inclusion of voice control into different applications as easy as possible. To this end, four C++ classes were developed. They allow the user to concentrate on designing the application's grammar and discovering the best combination of Nuance parameters, rather than reprogramming the same type of Nuance application. The end result of this effort is that the programmer only needs to instantiate explicitly one object, develop the grammar, and write the application specific code to handle the natural language results.

The four classes developed are the following:

- configClass
- recClientClass
- NLClass
- recognizerClass

The classes follow from the type of structures or "objects" used by the Nuance system. Each one will be described in detail.

1. Configuration - ConfigClass

The configClass is based in the CoronaConfig object used by Nuance. It is the basis of all the other functionality of the Nuance system. The CoronaConfig object is needed to call the API's to handle voice recognition (both client and server functions) and natural language processing. Hence, the configClass is needed by the recClientClass and the NLClass in order for those objects to be instantiated. The class definition of configClass is in Figure 13. The different constructors match the form of the Nuance API for creating a CoronaConfig object.

The configClass holds information that is held in the Nuance package that was compiled separately. This includes the number and names of top-level grammars that the recognizer can use to recognize speech, and what directory the current package is located in. It also knows whether NLP was built into the grammar. Hence, the configClass object is queried to see if an NLClass object needs to be instantiated to perform NLP.

```
class configClass : protected errorReportingClass {

public:

   //Constructors for the class. Implements the four ways Nuance provides
   //to initialize the recognizer client.
                  configClass(int *, char **, int = 1);
                  configClass(char *);
                  configClass(FILE *, char *, int = 1, int = 1,
                              CoronaConfig * = NULL);
                  configClass(int *, char **, char *, int = 1, int = 1,
                              int = 1, CoronaConfig * = NULL);

   //Destructor
                  ~configClass();

   //Default copy and operator = constructors
                  configClass(const configClass &);
   configClass&   operator=(const configClass &);

   //These functions get internal values of the config object.
   CoronaConfig  *getConfig() const;
   CoronaStatus   getStatus() const;
   char          *getPackageLocation() const;
   char          *getPackageID() const;
   char         **getGrammarNames() const;
   int            getNumberOfGrammars() const;
   int            isNLDefined() const;

   //May need to set the NLDefined private member, esp. from NLClass
   void           setNLDefined(int);

   //Print out all the config values. Useful for debugging.
   void           print() const;

private:
   void           makeGrammarNamesList();
   void           makePackageLocation();
   void           makePackageID();
   CoronaConfig  *config;
   CoronaStatus   status;
   char         **grammarList;
   int            numberOfGrammars;
   char          *packageLocation;
   char          *packageID;
   int            NLDefined;
};
```

Figure 13. Class definition of configClass

2. Recognition Client - recClientClass

The recClientClass, along with the Nuance RecClient object, is responsible for processing speech input. At its simplest, it can be used to record speech and have it played back. It can also accept a Nuance package so that it can recognize speech, the results of which go into a Nuance structure called RecResult. Through this structure, a user can get the text of what the recognition server thought was spoken. The recClientClass uses the Nuance RecClient API which handles the search for, and creation of, the recognition server. Hence it is an easier API to use, although more limited, than the RecServer API calls which a developer can use to make his own RecClient and server.

The recClientClass definition is in Figure 14. Again, the constructors match the format of the Nuance API calls. The only parameters that are not included are those that are member functions of the recClientClass, such as the RecClient pointer and the CoronaStatus variable. From the class definition, one can see that the recClientClass has several responsibilities; listen, record, playback, abort recognition, and for getting and setting Nuance system parameters. Which parameters can be retrieved and altered during runtime are found in the Nuance Developer's Manual.

However, one of the most important duties of the recClientClass is the registration of callback functions for Nuance system events. There are seven events that may occur. They are:

- CORONA_EVENT_PROCESS_DIED
- CORONA_EVENT_INIT_COMPLETE
- CORONA_EVENT_START_OF_SPEECH
- CORONA_EVENT_END_OF_SPEECH
- CORONA_EVENT_PARTIAL_RESULT
- CORONA_EVENT_FINAL_RESULT
- CORONA_EVENT_PLATBACK_DONE

The Nuance system is informed of events using the Unix select() command or by using the XtAppAddInput() in the X event loop to monitor a file to see if it is readable. If so, an event has happened which may be handled. In this application, calls were made using

```
class recClientClass : protected errorReportingClass {

public:
    //Constructors, operator =, and destructor
                recClientClass(configClass *, int = 60, XtAppContext = NULL);
                 recClientClass(const recClientClass &);
                 recClientClass &operator=(const recClientClass &);
    virtual     ~recClientClass();

    //Start, stop, set callbacks functions
    virtual void listen(char * = NULL, float = 3.0);
    virtual void startListening(char * = NULL);
    virtual void stopListening();
    virtual void abort();
    virtual void regCallback(CoronaEvent, RCCallbackFnPtr);

    //Utterance playback/stop functions
    virtual void playFile(char *, int = 60);
    virtual void playLastUtterance(int = 60);
    virtual void killPlayback();

    //Query functions and parameter setting
    virtual void getResults(int, void *) const;
    virtual RecClient *getRecClient() const;
    virtual void setParameter(int, int);
    virtual void setParameter(int, char);
    virtual void getParameter(int, int *) const;
    virtual void getParameter(int, char *) const;
    virtual int  isWaitingForEvent() const;

    //Recognizer results, open to the public
    RecResult   *resultsPtr;

protected:
    //Set up X callbacks in C++ environment
    static void  suddenDeathCB();
    static void  processEventsCB(recClientClass **);
    virtual void processEvents(RecClient *);
    virtual void setupSuddenDeath();

    virtual void initRecClient(int, XtAppContext);
    CoronaConfig *configPtr;
    RecClient   *clientPtr;
    int          recFileDescriptor;
    int          waitingForEvent;
    CoronaStatus status;
    recClientClass * recClientClassObjectPtr;
};
```

Figure 14. Class definition of recClientClass

the recClientClass regCallback function, with static member functions of the voiceVkApp class passed in to handle the different Nuance events.

A current problem with the recClientClass which has not been solved yet is the requirement to pass an object pointer since static member functions do not have access to non-static member functions or data. With X-style programs, callback functions can at least pass a this pointer as the data parameter. Then inside the static member function, a non-static member function can be called using the this pointer. However, the Nuance callbacks have a set parameter list that pass information for the Nuance event that precludes sending the this pointer. The quick solution was to have the recClientClass use a global variable in its C++ code that could be used be the Nuance event processing routine to call member functions. The use of this global variable precludes the instantiation of more than one recClientClass object in a single application. While this may sound severe, more than one top-level grammar can be used in a package, and calls can be made to make another grammar the current grammar if needed.

3. Natural Language Processing - NLClass

The NLClass is used to process the results of the voice recognition process, or process a string, if NLP was defined in the current package. It uses the Nuance NLEngine and the NLResult structures to do the processing. The class definition of NLClass is in Figure 15. As with the other classes, the constructors follow the format of the Nuance API calls, except for eliminating data members from the parameter list.

The NLClass allows the user access to the NL slot names and values in a means designed to make printing and transmitting that data simple. The user can get slot names as a list of strings. It also allows the user to get individual slot values as either strings or integers. The full list of slot values can also be retrieved as a list of strings (regardless if the original value was an integer). The user should know that this data is returned in dynamically allocated memory, and it is the programmer’s responsibility to delete that

```
class NLClass : protected errorReportingClass {

public:
    //Constructors and destructor
             NLClass(configClass *);   //uses config object
             NLClass(char *);          //uses package directory name
             ~NLClass();

    //Gets either all the slot names, or individual slot
    //names starting at 0.
    char **  getSlotNameList();
    char *   getSlotName(int);
    int      getLongestSlotNameLen();

    //Gets either all the values for the slots, or individual
    //values starting at slot 0, or by giving the slot name.
    char **  getSlotValueList();
    char *   getSlotValue(int);
    char *   getSlotValue(char *);
    int      getNumberOfSlots(){return numberOfSlots;};

    //Makes the NL engine interpret the results of the recognition
    //or interprets plain text.
    void     interpret(RecResult *);
    void     interpret(char *);

protected:
    int      countNumberOfSlots();
    char **  buildSlotNameList();
    CoronaConfig *configPtr;
    NLEngine *nlEngine;
    NLResult *nlResult;
    char **  slotNameList;
    int      numberOfSlots;
    int      longestSlotNameLen;
    CoronaStatus status;
};
```

Figure 15. Class definition of NLClass

memory when no longer needed. Additionally, the Nuance ability to handle structures of data is not currently supported in the class.

4. Recognition System Container - recognizerClass

The recognizerClass is meant to be a simple container class which can handle the instantiation and destruction of the three different classes it contains, the configClass, the recClientClass, and the NLClass, on an as-needed basis. The individual objects are public

members which are accessed directly through the recognizer object. The class definition of the recognizerClass is in Figure 16. The constructors have all the information needed to

```
class recognizerClass {

public:
    //Constructors - match items needed by configClass, recClientClass
    //and NLClass constructors
    recognizerClass(int *, char **, int = 1, int = 60, XtAppContext = NULL);
    recognizerClass(char *, int = 60, XtAppContext = NULL);
    recognizerClass(FILE *, char *, int = 1, int = 1, CoronaConfig * = NULL,
                    int = 60, XtAppContext = NULL);
    recognizerClass(int *, char **, char *, int, int, int,
                    CoronaConfig *, int = 60, XtAppContext = NULL);

    //Destructor
    ~recognizerClass(){};

    //Action methods for dealing with internal/external objects
    void changeConfig(FILE *, char *, int = 1, int = 1);
    void changeConfig(int *, char **, char *, int, int, int);

    //Objects needed to do the work
    configClass    config;
    recClientClass client;
    NLClass        NLProcessor;
};
```

Figure 16. Class definition of recognizerClass

instantiate the three member objects. The constructors each check to see if a package has been passed in, and if that package has NL defined by instantiating the configClass object first, then querying it. If NL is not defined, then that object is not instantiated. Hence, it is important for the user to query the configClass object to see if NL is defined. Otherwise, the program can easily crash if the non-existent NLClass object is accessed.

This class also has the ability to destroy its objects and create new ones in response to the overloaded changeConfig member functions. These two functions model the two CoronaConfig API calls that allow the changing of the config structure during runtime. However, these classes cannot simply change themselves. The configClass, instead of passing out the real address of the Nuance CoronaConfig object needed by many API calls

and the RecClient and NL objects, makes copies of its CoronaConfig object. Hence changing the configClass's CoronaConfig object does not change any of the copies that might have been created during program execution. Therefore, those objects are destroyed and new ones are created. The user must ensure that outdated copies of the CoronaConfig object are destroyed or updated.

C. VOICE CONTROL PANEL

The Voice Control Panel gives the feedback the user needs in order to see if he has been understood correctly, to see what he has said in the past, and to help him see what stage of voice processing the system is in. The Voice Control Panel (VCP) is shown in Figure 17. It is made up of the following components:

- Recognition Display Panel
- NL Display Panel
- Status Display Panel
- Menu Structure
- Editor

These components form an IDE which can help a developer more easily develop a voice interface by putting Nuance tools within quick reach, while at the same time providing feedback to the user concerning the results the system is producing.

1. Recognition Display Panel

The Recognition Display Panel (Figure 18) supplies two pieces of information, the current recognition result that the user is interested in, located on the top textfield, and a history list of prior recognition results. The user can double click with the left mouse button to see a prior result in the current command field, and have its NL interpretation displayed in the NL Display Panel, described below.

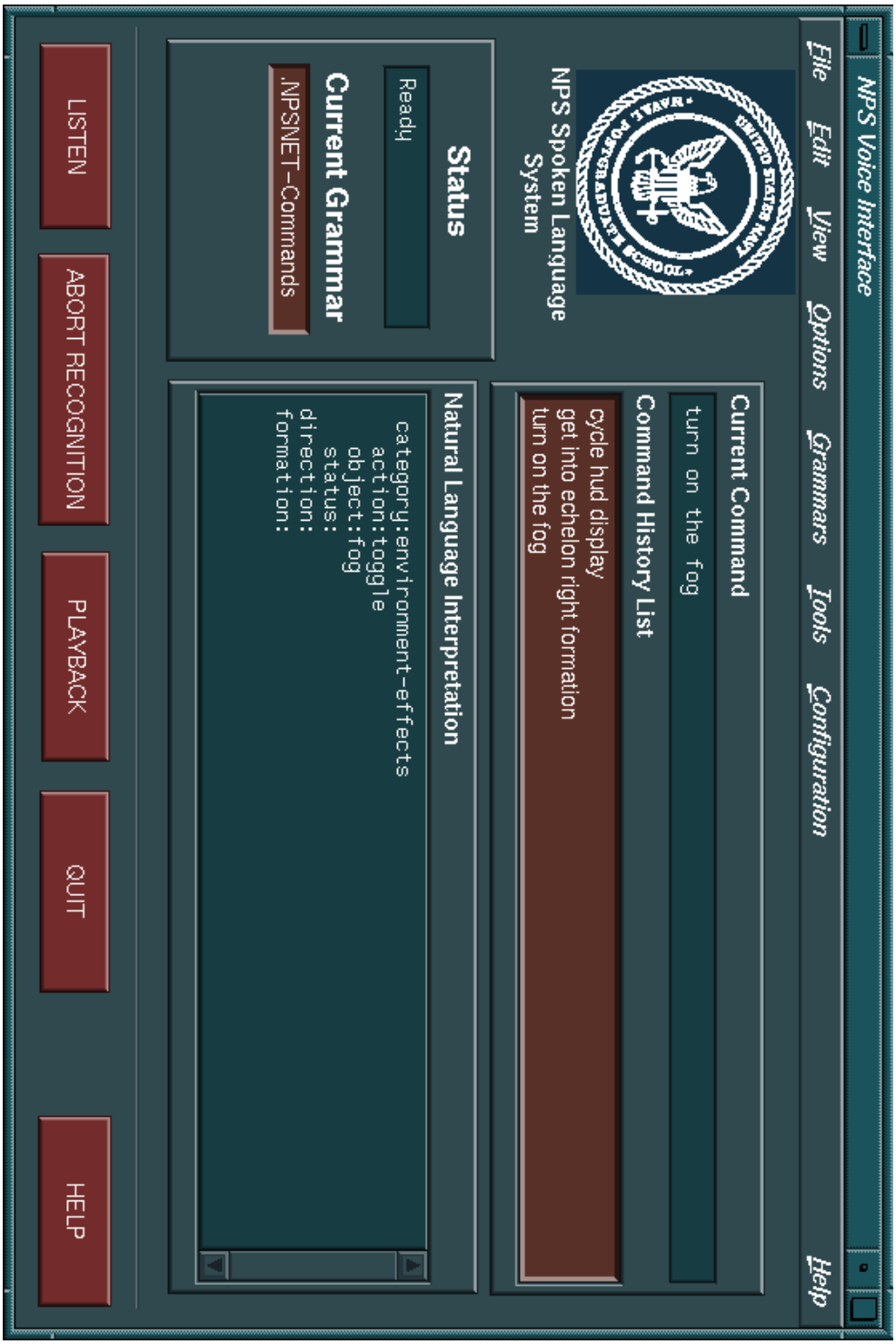

Figure 17. Voice Control Panel

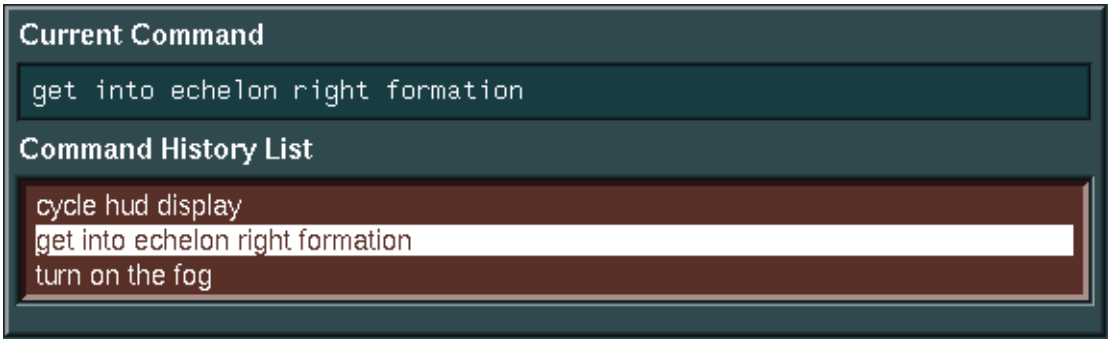

Figure 18. Recognition Display Panel

2. NL Display Panel

The NL Display Panel (Figure 19) shows all the slots defined in the current package in a right justified format in a scrolled text window. The NL interpretation of the current recognition result is displayed, or the interpretation of a prior result which has been picked in the Recognition Display panel's command history field. Most interpretations will not fill all the slots that have been defined.

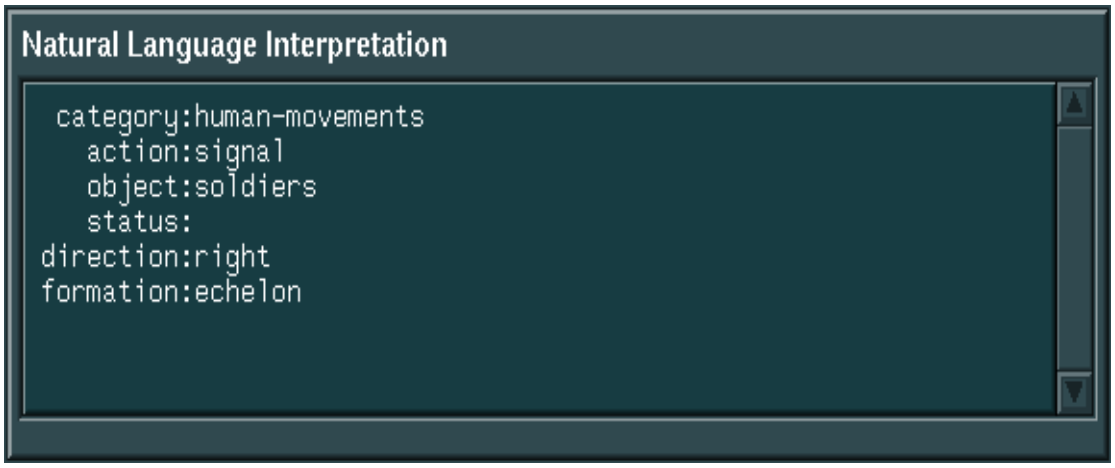

Figure 19. NL Display Panel

3. Status Display Panel

The Status Display Panel (Figure 20) gives feedback to the user about what is going on within the system, and it allows the user to choose which top-level grammar he wants to use. The status text field will give the following responses:

- Initializing
- Ready
- Listening
- End Listening
- Processing

The Grammar list field either displays “None Loaded” if no package has been loaded, or it gives a list, one entry viewable at a time, of the top-level grammars in the package. The user can select one of the grammars by simply double clicking with the left mouse button on his selection.

Figure 20. Status Display Class

4. Menu Structure

The menu structure gives access to much of the functionality of the SLU System. The following outline gives the menu entries, and indicates which choices have not yet been implemented.

- File: Fully completed. Used to call the editor, or exit the IDE.

  New: Brings up an editor window.

  Open: Select a file from a file selector dialog.

  Close: Iconifies the VCP.

Exit: Quits the application.

- Edit: None of the items in this menu has been implemented.
- View: All items have been implemented.

  Clear Current Command.

  Clear Command History.

  Clear NL Interpretation.

- Options: Half implemented.

  Network - Opens/closes up the IDU network, depending on its status.

  Continuous Listening: Not yet implemented.

- Grammar: Not fully implemented.

  Text View: Loads the current grammar file into an editor window. It assumes that the grammar file has the same name as the directory in which the package is located, and that it has the ".grammar" extension.

  Graph View: Not yet implemented.

  Load Package: If the package was not loaded during initialization, or if a new package is needed, the directory in which the new package is located is entered into a prompt dialog.

- Configuration: Not yet implemented.
- Tools: Not yet implemented.
- Help: Not yet implemented.

5. Editor

A simple editor (Figure 21) with search/replace functionality was added to allow the user to quickly view files, make new grammars, or view the currently loaded grammar file from within the voiceApp application. The editor can be called from the File and Grammar menu items on the VCP main window. It can fully search and replace strings, and load and save files as one would expect. Once it is no longer needed, it can be exited and its resources will be freed to the system.

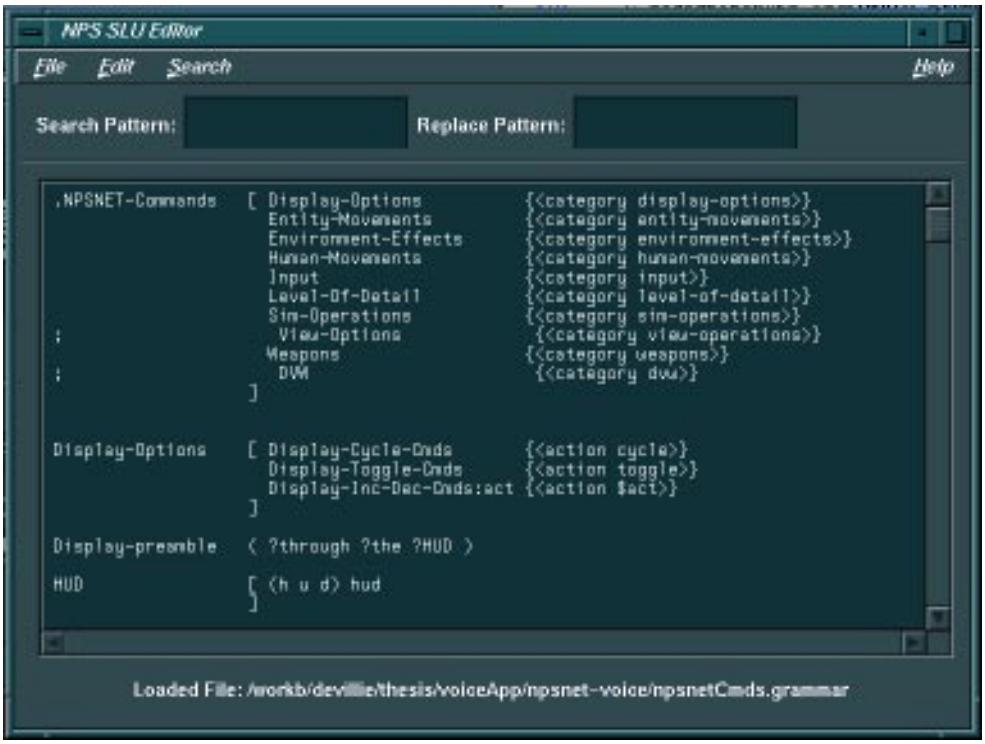

Figure 21. Editor window

D. NETWORK INTERFACE

The Spoken Language System is a network capable system for several reasons. First, it is made to interface with VR environments such as NPSNET which are themselves network capable. This allows the voice interface to be used with different workstations running NPSNET or another voice capable application, and the workstations can be in geographically different locations. Second, voice processing usually requires a good amount of computing power. VR applications such as NPSNET work best when using multiprocessor computers to support both the intensive graphic throughput needed (30 frames per second) and the management of possibly hundreds of entities distributed across a network. In this situation, it is best to run the voice software on a non-graphics related workstation. Third, using the network gives the voice interface the ability to serve multiple computers running voice enabled applications at the same time.

1. Overall Structure

The general network structure of the SLU system and NPSNET can be found in Figure 10. The system needs a “client” and “server” network manager. The “client” net manager takes the data from the NLClass object and puts it into a predefined C/C++ structure. This structure can then be put into a packet and sent across the network. The “server” net manager periodically reads the network to see if any packets of the structure type sent from the client have been recieved. If so, it reads them and translate the NL data into the necessary action.

2. NPSNET Network and Interface Handling

NPSNET is a Distributed Interactive Simulation (DIS) compliant virtual environment. This means that players (entities), actions (explosions, weapons fire, collisions, etc.), and environmental effects (wind, electromagnetic radiation, etc.) are processed and managed using a consistent interchange of predefined structural units called Packet Data Units (PDUs). These PDU’s contain the data necessary to determine vehicle type, orientation, and speed, weapon interaction, etc. In this way, users on different workstations interacting in the same simulation will see the same entities and actions from their different perspectives.

Each user in the simulation is in control of a vehicle or “entity.” The user has several ways to interact with the simulation: keyboard, joystick, flight control sticks (FCS), the omnidirectional treadmill, mouse, spaceball, etc. These input devices can control things such as entity movement and actions, NPSNET screen displays, environmental effects such as fog and clouds, and information display. Currently, only the keyboard allows access to all the functionality of NPSNET, and as such it is always active as an input device in an NPSNET simulation, regardless of any other input device used.

Since DIS is a predefined protocol, another method, the Information Data Unit (IDU) network, was created to send varying information to NPSNET simulations which is not defined in DIS. The IDU net manager works in parallel to the DIS net manager.

NPSNET defines different IDU structures that hold varying amouts and types of information. Such structures are used to pass information and commands in several NPSNET applications, such as the submarine [BACO95] and OOD trainers [STEW96]. The SLU system also uses the IDU net to pass the NLP results to a voice module inside NPSNET which takes the data and translates that into an action.

a. General NPSNET Operation

NPSNET uses a main simulation loop in which each pass through the loop is one frame drawn in the VR display. During this loop, NPSNET performs the following actions (not an exhuastive list):

- Checks the network for new DIS packets.
- Parses appropriate DIS packets for new entity and environement information, such as orientation, speed, status (alive or dead), etc.
- Queries all input devices for new data by using a centralized input-manager.
- Updates entity and environemental states as appropriate.
- Draws the scene.
- Starts the loop over again until it gets the exit signal.

Each of the separate parts of NPSNET which control the entity, HUD display, information display, environmental effects, etc., poll the input manager to see if specific input has been received. For example, the entity, no matter if it is a tank or helocopter, will check to see if an ‘a’ has been pressed. An ‘a’ keypress causes the entity to accelerate. The input manager will then query all the input devices that are active whether they have received an ‘a’ keypress. How this query is interpreted is dependent on the input device. A query to the FCS about an ‘a’ keypress may have that device handler see if actually the throttle has been pushed forward. In this way, the interface to all interface devices, whose handling objects are derived from a base input-device class, is the same.

Two points should be obvious about NPSNET input handling. First, the polling process means that there is no one place in NPSNET in which the code providing the functionality of the system can be seen. It is not possible to go to the input-manager, the main simulation loop, nor any of the device classes and see what the ‘a’ keypress does, nor

which part of NPSNET is effected by it. Second, any new input device derived from the base input-device class needs to translate its input into the keypresses that the parts of NPSNET will look for.

b. Information Data Unit (IDU) Network

The IDU net is used in NPSNET within some of the entity classes to receive special control or coordination information which that entity may need. For example, the OOD trainer [STEW96] instantiates an IDU net manager within the entity object of the ship. When NPSNET calls the ship entity's move function to update its position, the function checks the IDU net for any IDU packets of the OodToNPSNETIDU type. If any are found, the ship's internal state which controls speed, rudder action, direction, etc. are updated. An NPSNETToOODIDU type packet is then sent out so the ship's control panel can update its GUI, providing feedback to the user.

Currently, the main header files that define the workings of NPSNET's IDU net are idu.h and idunetlib.h. Both can be found in the src/communication/include directory. The idu.h file contains the definition of the idu structures used in different applications of NPSNET (Figure 22).

```
/* Types */
typedef unsigned charIDUType;

#define OtherIDU_Type(IDUType)0
#define Test_Type(IDUType)1
#define ViewpointControl_Type(IDUType)2
#define SS_To_Jack_Type(IDUType)101
#define Jack_To_SS_Type(IDUType)102
#ifndef NOSUB
#define NPSNET_To_SUB_Type              (IDUType)103
#define SUB_Helm_To_NPSNET_Type         (IDUType)104
#define SUB_Ood_To_NPSNET_Type          (IDUType)105
#define SUB_Weaps_To_NPSNET_Type        (IDUType)106
#endif // NOSUB

#define NPSNET_To_SHIP_Type             (IDUType)107
#define SHIP_Ood_To_NPSNET_Type         (IDUType)108
#define VoiceApp_To_NPSNET_Type         (IDUType)109
```

Figure 22. IDU type definitions

The idunetlib.h file defines the IDU_net_manager class. This class provides the mechanism to do the following:

- Open, close, and monitor the network.
- Specify whether broadcast or multicast transmissions will be used
- Specify specific ports and/or addresses to use.
- Read and write previously defined IDU structures.
- Specify which types of IDU structures to read from the network. All others are ignored.

3. SLU System IDU Net Management Classes

The voiceNetManagerClass and the voiceModuleClass both instantiate an IDU_net_manager class object which handles the actual opening, reading, writing, and closing of the network. A new IDU structure was defined for this voice control application, as seen in Figure 23. The structure has the standard IDU header which holds what type of

```
//The following struct defines the idu used by voiceApp for
//voice control of NPSNET or any voice enabled app.
typedef struct {
  IDUHeader    header;
  char         appName[15];  //Name of app that this IDU is for
  char         data[210];    //Holds all data for specifying cmd
  ulong        space_holder;
} VoiceAppToNPSNETIDU;
```

Figure 23. New IDU structure for SLU System

structure this is. This information is used by the IDU_net_manager object to see if it should take the packet from the network queue. The appName field was entered so that an application with different functional areas could send this structure to the appropriate part of the program, or interpret the contents in a different way depending on the string entered in the appName field. The data is contained in a 210 byte wide field. How this field is actually broken up to contain information is meant to be application dependent, with the appName field indicating to the application what should be done.

For example, the SLU system uses the appName string “NPSNET” and assumes a structure of twelve, 15-character, null terminated string fields into which values from the Nuance NL slots are placed in the order they were defined. Basically, how the values are seen in the NL display Panel of the Voice Control Panel is how they are arranged in the structure. This includes null values. Hence, the structure could contain mostly wasted space. This was seen as a valid trade-off for the added flexibility of not being restricted to a predefined structure which may not fit current application needs.

a. voiceNetManagerClass

The voiceNetManager is used with the SLU system to take an NL interpretation and convert it into an IDU structure which is then written to the net. The voiceNetManagerClass object is instantiated upon request of the user. The user toggles the network under the menu “Options”. It will not be instantiated in any case if natural language use was not defined within the Nuance package currently loaded.

Once instantiated, the forming of the IDU packet and its writing to the net are automatically accomplished through calls within the finalResultCB function within the voiceVkApp class. The voiceNetManagerClass object first checks to see if the NL results are empty because of a rejected recognition. If not empty, it gets the NL values as a list of strings, and puts each string into its corresponding field. It then writes the IDU structure onto the net.

b. voiceModuleClass

Unlike past uses of the IDU net within NPSNET [BACO95] [STEW96], the voiceModuleClass was designed to work inside the main simulation loop of NPSNET just like the DIS_net_manager. It monitors the network for voice type IDU packets, reads them, and translates them into the corresponding keyboard sequence. It then supplies the keyboard class object the key, and lets the normal processing of input cause the desired action to execute.

The following is a detailed description of the processes inside NPSNET that makes voice control work. After querying the Initialization Manager to see if voice control was specified, the main NPSNET routine instantiates the voiceModuleClass if the query was positive. Directly after NPSNET reads and parses incoming DIS packets using the DIS_net_manager, the voiceModuleClass object does its processing. This processing includes:

- Read the net for VoiceAppToNPSNETIDU type packets which are the only type of IDU packet that the voiceModuleClass is concerned about. Therefore, it will not interfere with the other applications within NPSNET that use the IDU net.
- Parse the IDU packet in a large but straightforward C-style switch statement that results in the corresponding key sequence needed to execute this action as if it were being requested through the keyboard.
- Place the new key, and any modifying keypresses such as the Control or Shift keys, into the arrays in the keyboard class object handling the physical keyboard.

After the voiceModuleClass finishes its processing, NPSNET then processes all inputs. The individual sections of NPSNET start querying the input-manager to see if the “keys” that affect them have been pressed.

The keyboard class object handles the keyboard and mouse events that it receives as X system events. It is constantly collecting and processing these events. A keypress is handled by the use of two arrays which have an element for each key that can be pressed on the keyboard. One array, the keyPressCounter array, records how many times an individual key has been pressed. The other array, the lastKeyPressCount array, holds the number of times the key was pressed since the last time inputs were processed. By comparing the two arrays, NPSNET can determine if a specific key has been pressed since the last time it processed inputs. The keyboard class object also maintains a variable called keyState whose value is modified through the use of masks. When a modifying key, such as a control or shift key, is pressed, the keyState variable is masked with the appropriate value. Hence, NPSNET can also query the keyboard object about control, shift, and alt key

presses. When those keys are released, the mask for the key released is removed from the keyState variable.

In order to let the voiceModuleClass change the keyPressCount array, the keyboard class was modified to make the voiceModuleClass a friend. The voiceModuleClass could then access that private member. Also, when the input-manager is first instantiating the different input-device objects, it puts the address of the new keyboard class object into a global variable that the voiceModuleClass can use to access the keyboard object's arrays.

4. Alternative Solutions and Problems

Two other alternatives were considered to implement voice control in NPSNET. The first was to create a small, separate application running on the same machine as NPSNET. Then by using the X Window system, it would get the needed information about the NPSNET window and display. The application would then create the appropriate XEvent for a specific keypress, and send that event into the NPSNET window. The X Window system would process it as any other keypress, as would NPSNET. No modification of the NPSNET code would be necessary. Changes to this X program would only be necessary if the keyboard control interface were changed. However, I could not find a way to make this procedure happen. It should be possible to do, since other programs, such as SpeechManager by SGI, query the X system to see what window has input focus so they can determine if it has a vocabulary ready for that specific application.

The second method is to make the voiceModuleClass a derived class of the input-device class. It could then be queried by the input-manager just like any input device. However, NPSNET only allows two devices, the keyboard device and another derived input-device class, to be available at any one time. To keep the current structure, only the keyboard and the voice interface would be usable if voice was selected as the second input device. Voice was meant to augment the other input methods, especially when using devices such as the FCS.

The way NPSNET handles the keyboard is open to race conditions between the user and the keyboard which the current implementation of voice control does not help. The handling of keyboard events is based on the assumption that the key press and key release of modifying keys does not happen in between input processing events, i.e., within the period of one frame. For example, the up arrow for an air vehicle will make it increase its pitch. The Shift-up-arrow combination will have it gain altitude straight up. If NPSNET processes the shift-up-arrow event after the shift has been released, then the keyState variable will have no record of the shift key press. Therefore, instead of going straight up, the pitch will increase. If NPSNET is producing a high frame rate, this problem should not appear. Our fingers don't move that fast.

The current implementation now adds a race condition between the keyboard and the voice input. It is possible for the keyboard and the voiceModuleClass to add characters to the keyPressCount array almost simultaneously. This is not serious if no modifying characters were used. The different parts of NPSNET will query the input-manager about specific characters. However, if the keyboard or voiceModuleClass adds a modifying character which the other one does not want, that modifying character will still be applied to all characters. Again, with a high frame rate, this problem should not be noticeable. There is no fix to these problems since they originate from the design decisions made in constructing NPSNET.

VI. GRAMMAR DEVELOPMENT

Grammar development is one of the most crucial elements for a voice interface. The end-user will not typically know what the grammar is, nor exactly what one can say. Ideally, one should not have to know. This means that the developer must try to develop a grammar that will handle the most common words and grammatic structure that the prospective end-user will use. One way one can do this is through observing people working in the target environment. This method can be used for entity movement commands, and application specific actions. However, there is no current environment in NPSNET where people try to talk to their computer to control the display, VE view, or environmental options.

A. GOAL AND APPROACH

The goal of grammar development for the voice interface in this thesis is to have a one-to-many mapping of keyboard key-sequence commands to voice utterances that can be flexibly varied through the use of natural language processing. This eases the memory requirements of the user in using the full functionality of the NPSNET interface.

Two major approaches were taken to develop an initial grammar. As mentioned above, certain keyboard command categories, such as entity movement and human movement, are often spoken about during the use of NPSNET or are related to real-world actions. Other categories are not as obvious. For these commands, the NPSNET User's Guide [CSD96] was used to give an example of possible language a user may use to execute that command verbally.

Once the initial grammar is constructed, several tools included in the Nuance system were used to refine the grammar and the different Nuance parameters that can effect the accurracy and recognition latency of the system. These included a tool to randomly generate sentences using the grammar, tools to prerecord individual utterences, and a batch processing program that gives statistical results.

B. KEYBOARD COMMANDS

The keyboard is one of the main input devices for NPSNET. All simulations in NPSNET can have one or two input devices, one of which is always the keyboard. Therefore, most of the functionality of NPSNET can be accessed through the keyboard commands. These commands are categorized into ten functional groups:

- Display Options (Table 3)
- Entity Movement (Table 4)
- Environmental Effects (Table 5)
- Human Figure Movement (Table 6)
- Input (Table 7)
- Level of Detail (Table 7)
- Simulation Operation (Table 8)
- VE View Options (Table 9)
- Weapons (Table 10)
- DVW (Table 11)

Since the keyboard is central for input, the voice interface was designed to mimick the keyboard commands available. The grammar developed translates the specific actions available with the keyboard into phrases the user can speak to execute the corresponding keyboard command. Certain actions can be accomplished better using the keyboard, such as the repetitious striking of an arrow key to turn the vehicle. In order to get the same response, the voice interface would have to include the ability to continually send the turn command to the system, contain the grammar necessary to understand a “stop” command that halts the turning, and distinguish that voice command from a command telling the vehicle to stop moving completely. The current voice interface does not attempt to do this, but mimicks a single keyboard command occurrance per utterance.

C. PROBLEMS

A SLU system will have problems in addition to the ones found in text-based NLP systems. These include confusable words and recognition latency. Confusable words are those words with different meanings that may be used around the same place in the

Key	Description
{0}	Cycle HUD text colors.
{1}	Toggle HUD radar rotation mode. [Aim point]
{2}	Toggle HUD radar color mode.
{3}	Toggle HUD radar icon mode.
{m}	Toggle measuring system between metric/American. [Speed, Altitude gauges]
{n}	Narrow FoV. [Compass]
{o}	Toggle display of control measures.
{w}	Widen FoV. [Compass]
{B}	Toggle entity self-bounding box.
{Alt-F1}	When in full screen mode, bring window to foreground.
{Alt-F3}	When in full screen mode, bring desktop to foreground; then push windows to background, one at a time.
{Tab}	Cycle HUD transparency amount. [Status block]
{F1}	Toggle Performer statistics. [Performer statistics]
{Ctrl-F1}	Cycle through Performer statistics display options. [Performer statistics]
{F2}	Toggle Weapons block.
{F3}	Toggle Position block, Status block.
{F4}	Cycle through HUD options. Each key press adds another layer of graphic information to the display, until the cycle returns to none.
{F5}	Toggle texturing on/off (expect brief delay).
{F6}	Toggle wireframe on/off (expect brief delay).
{F7}	Cycle through multi-sampling levels for anti-aliasing.
{+}	Increase HUD radar range. [Status block]
{-}	Decrease HUD radar range. [Status block]

Table 3: Keyboard Display Options [CSD96]

Key	Description
{a}	Accelerate driven vehicle. [Speed gauge]
{d}	Decelerate (and/or reverse) driven vehicle. [Speed gauge]
{s}	Stop entity. [Speed gauge]
{LeftArrow}	When not paused, change roll to perform left bank (air vehicle) or turn left (ground vehicle).
{RightArrow}	When not paused, change roll to perform right bank (air vehicle) or turn right (ground vehicle).
{UpArrow}	When not paused, increase vehicle pitch (air vehicle only).
{DownArrow}	When not paused, decrease vehicle pitch (air vehicle only).
{Shift-UpArrow}	Increase altitude (air vehicle only). [Altitude gauge]
{Shift-DownArrow}	Decrease altitude (air vehicle only). [Altitude gauge]
{Insert}	Reset vehicle direction to view direction.

Table 4: Keyboard Entity Movement Commands [CSD96]

grammar which sound similar. The system could make more mistakes in such areas depending on factors that normally affect recognition, such as how clearly the user spoke the commands, background noise, and the influence of surrounding words and how this can affect the recognition system's processing. This last point highlights the fact that confusability is not just related to how closely words sound alike, but also where they are used. Recognition latency is related to all the factors that can affect the processing speed as described in Chapter II. These factors include grammar complexity, perplexity, and voice processing factors that Nuance allows to be controlled, such as the number of frames into which a voice signal is divided and the search breath of the Viterbi algorithm used by Nuance to find a best match between the grammar and the spoken input.

Key	Description
{c}	If not DVW, toggle clouds on/off.
{f}	If not DVW, toggle fog on/off.
{g} or {j}	Increase cloud level.
{h} or {k}	Decrease cloud level.
{t}	Cycle time-of-day lighting effect.
{u}	Decrease cloud thickness.
{y}	Double cloud thickness.
{Q}	Toggle quiet mode.
{>} or {.}	Increase fog.
{<} or {,}	Decrease fog.

Table 5: Keyboard Environmental Effects Commands [CSD96]

Key	Description	Key	Description
{e}	Set human to upright position.	{Ctrl-K}	Person move left.
{G}	Reset sight.	{Ctrl-N}	Person echelon right.
{H}	Left rifle sight.	{Ctrl-O}	Person echelon left.
{J}	Down rifle sight.	{Ctrl-P}	Person salute.
{K}	Up rifle sight.	{Ctrl-Q}	Person close up.
{L}	Right rifle sight.	{Ctrl-R}	Person vee.
{Ctrl-A}	Person forward.	{Ctrl-S}	Person halt.
{Ctrl-D}	Person point to.	{Ctrl-T}	Person column.
{Ctrl-F}	Person down.	{Ctrl-U}	Person wedge.
{Ctrl-G}	Person speed.	{Ctrl-V}	Person j-line.
{Ctrl-J}	Person move right.	{Ctrl-W}	Person open up.

Table 6: Keyboard Human Movement Commands [CSD96]

Key	Description
{i}	Change available input device. [Status block]
{Shift-Enter}	Reset resting stick position.
{x}	Increase filter range. [Status block]
{z}	Decrease filter range. [Status block]

Table 7: Keyboard Input and Level-of-Detail Commands [CSD96]

Key	Description
{r}	If not DVW, reset terrain and static databases.
{Ctrl-Backspace}	Toggle entity invulnerability. [Status block]
{Esc}	Exit NPSNET.
{Pause}	Toggle pause/resume. [bottom middle] (Closing the window to an icon will also pause NPSNET.)
{PrintScreen}	Save NPSNET image to /tmp/nps_screen.#.rgb, where # is an incremental numbering of image files. (Save directory can be overridden with NPS_PIC_DIR. See UG: 3.3. Environment Variables.)
{F9} to {F12}	Reset vehicle position and orientation to specific points in the VE. See DFS: 1.17. Transport File for reconfiguration guide. When paused, these keys will save the current position as a new transport location and update the Transport File accordingly.
{Pad+}	Increase radar range. See {F4}.
{Pad-}	Decrease radar range. See {F4}. (Pad minus key.)
{Shift-Backspace}	Reset entity state to alive/healthy.
{?}	Print current networking variables--port number, networking mode, etc.--to the shell NPSNET was started from.

Table 8: Keyboard Simulation Operation Commands [CSD96]

Key	Description
{6}	Toggle pilot/bird's-eye view.
{Delete}	Reset view position to vehicle position.
{PageDown}	When in bird's-eye view, increase distance (zoom out).
{PageUp}	When in bird's-eye view, decrease distance (zoom in).
{Pad5}	Reset view direction to vehicle direction. [Compass]
{PadUpArrow}	Rotate view up.
{PadDownArrow}	Rotate view down.
{PadLeftArrow}	Rotate view left. [Compass]
{PadRightArrow}	Rotate view right. [Compass]
{UpArrow}	When paused, rotate view forward.
{DownArrow}	When paused, rotate view backward.
{LeftArrow}	When paused, rotate view left.
{RightArrow}	When paused, rotate view right.
{Ctrl-PadUpArrow}	In bird's-eye view, rotate view up.
{Ctrl-PadDownArrow}	In bird's-eye view, rotate view down.
{Ctrl-PadLeftArrow}	In bird's-eye view, rotate view left.
{Ctrl-PadRightArrow}	In bird's-eye view, rotate view right.

Table 9: Keyboard VE View Options [CSD96]

1. Confusable Words

The initial grammar had several areas where confusable words became a problem. These problems were related to the similar sounds of the words and the structure of the grammar itself. Examples of the former problem are the words "increase", "decrease", "accelerate", and "decelerate." The word "decrease" was confused for the word "increase" approximately 50% of the time. Other examples include "speed up" being confused for the word "stop", and an unintuitive mistake of "left" being confused for "up."

Key	Description
{4}	Toggle targeting. [Status block]
{q}	Cycle munition-specific field. [Status block]
{v}	Enable visual missile mode. [Weapons block]
{Home}	Fire primary weapon. [Weapons block]
{End}	Fire secondary weapon. [Weapons block]
{SpaceBar}	Fire tertiary weapon. [Weapons block]

Table 10: Keyboard Weapon Commands [CSD96]

Key	Description
{c}	Send cloud PDU test.
{f}	Set haze on.
{p}	Toggle DVW print switch.
{r}	Clean clouds and delete plumes.
{C}	Enable clouds. (Default is off.)
{D}	Toggle dust. (Default is off.)
{SpaceBar}	Create new flare.

Table 11: Keyboard DVW Commands [CSD96]

The initial grammar was made too flexible, and this led to misinterpretations. For example, the display commands dealing with the HUD had the option of mentioning the HUD as an adjective. Actually, most adjectives were made optional in the initial grammar. If this adjective phrase was used in the command “cycle through the HUD text colors,” the system reached a correct recognition result. There were enough words to distinguish it from other commands. However, if the optional words were not used, then phrases such as “cycle colors” or “cycle the colors” were interpreted as “column.” “Column” is a human movement command and has nothing to do with the HUD. But the words sound similar.

Two things were done to reduce the error rate from confusability. First, confusable words were taken out of some of the rewrite rules. An example of a deleted word was “decelerate”, as in “decelerate the tank.” It was assumed that this would be an unlikely word to be used in practice for slowing down a vehicle. Of course, it should be pointed out that there is no empirical evidence that says that this assumption is true. The second action taken was to limit the flexibility of the grammar to reduce the error rate. Many of the uses of adjectives and articles (i.e., the, a, an) became required in the updated grammar rather than optional. Both the initial and updated grammar are available for comparison in APPENDIX A.

2. Recognition Latency

Control interfaces are usually required to have a low recognition latency as well as a low error rate [MARK96]. The SLU system of this thesis tries to minimize both the latency and the error rate. This means that compromises have to be made within the system as explained in Chapter II.

One of the most effective means to reduce the latency of the SLU system was to choose the most appropriate acoustic model supplied by the Nuance system. As explained in Chapter IV, Nuance comes with three acoustic models which have trade-offs between accurracy and speed. Tests were done to see how two of the models, the genome (gen) and the phonetically tied mixture (PTM) models, performed as different Nuance parameters were varied.

There are two major parameters in the Nuance system that can be used to vary the accurracy/speed trade-off. These are the rec.Pruning and the rec.SkipObsFrames parameters. The rec.Pruning parameter affects how wide of a search the recognition system conducts to find the best match. The Nuance system is based on Hidden Markov Models which the system uses to create a stochastic network representing the entire grammar. The system, during recognition, uses the Viterbi algorithm to find the best path through the HMM network which results in its best guess about what was said. If the rec.Pruning

parameter is set very low, the search area becomes very small, and the algorithm becomes very greedy [SHAH96]. The best choice is taken at each turn through the network, with very little backtracking taking place. The danger here is that a mistake with the first couple of words can cause the whole recognition to fail. This effect can be seen by purposely making errors in different parts of the input statement. Making the rec.Pruning parameter large may increase the accuracy, but it increases the search space and the processing time. Making it small may decrease the time, but lower the accuracy.

The rec.SkipObsFrames parameter sets a threshold value that determines how a recognition hypothesis will be evaluated. The system initially makes a hypothesis about what was said and assigns it a probability number. Those hypothese with a probability number higher than that set by the rec.SkipObsFrames parameter will be processes using a more computationally expensive method than if it had a number below the set threshold value.

D. RESULTS

In order to optimize the system for both reduced recognition processing time and error rate, tests were done that varied the two parameters mentioned above. These tests were repeated using the different acoustic models included with the Nuance system.

1. Test Procedures

In order to test the effects of varying parameters and acoustic models, the batchrec tool included in the Nuance system was used. It allows batch processing of pre-recorded voice sound files and gives processing statistics like those in Figure 24. The most important of these for this thesis was the total average processing time. With included transcripts of what the voice files actually contain and their NL interpretation, batchrec will also produce statistics on the individual and cummulative accuracy of the recognition and NL systems.

In order to do this processing, the following steps were taken:

- Use the random-generate tool in Nuance to generate a set number of sentences using the updated NPSNET grammar. Two hundred sentences were generated.

```
SENTENCE: 198

SENTENCE: 198 pass1
HYP:  CYCLE THE PERFORMER STATISTICS
PROB: -11754
PROB_PER_FRAME: -44
PFSG_REALS: 45 active, 33 ends, 68 starts, 41 saved, 7 pruned (16.2%), 18 bt
PFSG_NULLS: 6 active (11.7%), 8 ends, 22 starts, 2 levels
PFSG_REJECTS: 8 active (18.2%), 0 ends, 13 starts (17.0%)
TIMES: 6.57 secs (6.01p 0.10g) 2.50xRT (4.58u 0.03s 1.75xcpuRT)
TOTAL_TIMES: 5.37 secs (4.97p 0.08g) 2.87xRT
GAUSS: -1.40167e+08
GAUSS_PER_FRAME: -532956
NUM_FRAMES: 263
ACTIVE_GAUSSIANS: frame_single_feature 105.0
GAUSSIANS_STARTED: frame_single_feature 2502.4
GENONES_PER_FRAME: frame_single_feature 78.2
```

Figure 24. Sample batchrec Output

- Create two different packages using the updated NPSNET grammar that use the Genome and PTM acoustic models.
- Create a file that lists all the sound files that will be used with batchrec.
- Use the Xwavedit tool (Figure 25) to record the reading of each generated sentence into its own sound file.
- Create two files. One will have the exact transcript of what was spoken in each sound file. Each line will have the name of the sound file and the sentence. The other file will have the NL interpretation. Each line will have the name of the sound file followed by the those NL slots that should be filled after NLP and the slot value.
- Run the batchrec program on both packages. Use the Unix script command to record the batchrec statistics that are sent to the screen, and redirect all other output into a separate file for each run. Vary the rec.Pruning and rec.SkipObsFrames parameters for different runs through both packages.

2. Results

A comparison of recognition time, and the accuracy of the voice and NL processing using the Genome and PTM acoustic models and various values for the rec.Pruning and rec.SkipObsFrames can be found in Table 12 and Table 13. From the data collected, it can

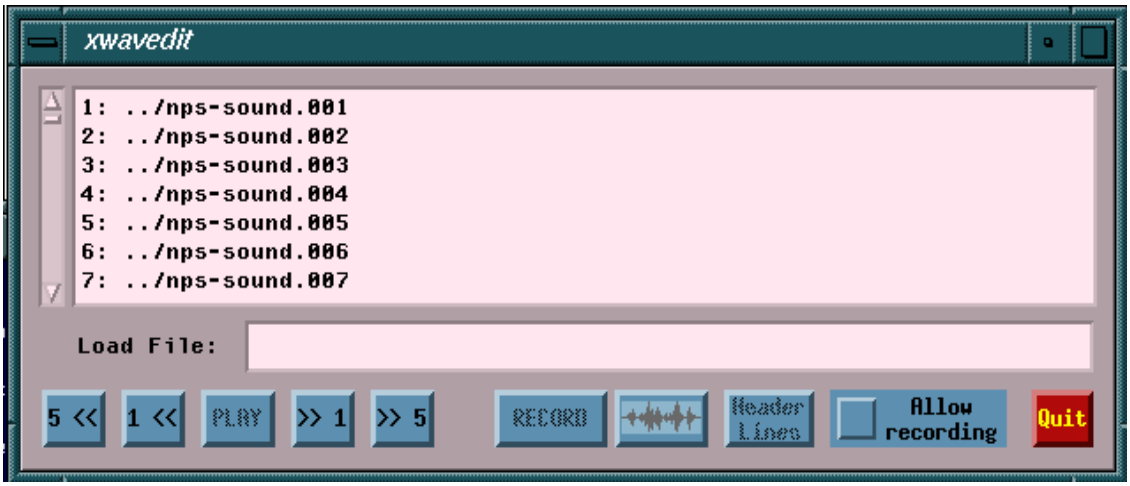

Figure 25. Xwavedit Tool

Model	PTM				Genome			
SkipObsFrame	$1x10^6$	$7x10^5$	$5x10^5$	$3x10^5$	$1x10^6$	$7x10^5$	$5x10^5$	$3x10^5$
Avg. Rec. Time	2.55	2.78	2.56	3.78	5.38	5.22	6.30	6.57
Avg. Rec. % Error	21.2	21.2	21.2	21.2	21.2	21.2	21.2	21.2
Avg NL % Error	15.7	15.7	15.7	15.7	14.6	14.6	14.6	14.6

Table 12: Recognition Results using rec.Pruning = 800

be seen that the PTM acoustic model reduces the average recognition time from 43.0% to 50.3% compared to the genome model using rec.Pruning values of 700 and 800 respectively.

The data showing error rates was surprising. The recognition and NL error rates for both the PTM and the more accurate genome acoustic models were almost the same. Using a rec.Pruning value of 800, there was no recognition error rate difference between the two acoustic model, and the NL error rate was only 1.1% better using the genome model that approximately doubled the processing time. Using a rec.Pruning value of 700, the genome

Model	PTM				Genome			
SkipObsFrame	$1x10^6$	$7x10^5$	$5x10^5$	$3x10^5$	$1x10^6$	$7x10^5$	$5x10^5$	$3x10^5$
Avg. Rec. Time	2.36	2.46	2.47	2.45	4.27	4.26	4.29	4.27
Avg. Rec. % Error	21.2	21.2	21.2	21.2	21.7	21.7	21.7	21.7
Avg NL % Error	16.2	16.2	16.2	16.2	16.2	16.2	16.2	16.2

Table 13: Recognition Results using rec.Pruning = 700

value actual performed slightly worse with a 0.5% higher recognition error rate. There was no difference in the NL error rate between the two models. Also surprising was the fact that there was no change in error rates with the change in the rec.SkipObsFrames parameter.

Decreasing the rec.SkipObsFrames parameter had limited success. The greatest effects were seen when using the more computationally demanding genome model. Reducing the value from the default of 1,000,000 to 700,000 decreased the recognition time by 3.0% when using a rec.Pruning value of 800. Changing the rec.Pruning value to 700 had a much greater effect on the genome model's recognition processing time with an average reduction of 26.5%. Other than that, changing the rec.Pruning had little affect on recognition and NL error rates. The reduction of the rec.SkipObsPruning had a negative effect on recognition time.

Comparing the recognition and NL error rates, it is important to note that the NL error rate is less than the recognition error rate. While the Nuance system counts the addition or deletion of an article of speech as an error, this had no affect on the meaning of the sentence. Hence, using NLP not only adds flexibility to the system, but helps increase its accuracy. Reviewing the transcripts of the batch processing, the errors came out in the following groups:

- 23% of all recognition errors resulted from the deletion of an article or preposition that did not affect the NL result.

- 17% of all NL errors would result in no action of the voice control system. The entire phrase was rejected by the system.
- 16% of all NL errors were the result of the misinterpretation of the same input sentence. Repetition of sentences was allowed since the recognition can depend on how a sentence was spoken, not just on what was spoken.
- 21% of the NL errors would have resulted in an action by the voice control system that was not related to the same category of action requested by the user. This means that approximately 3% of all requests will result in an action completely unrelated to the action desired.

The results show that for the NPSNET environment and the grammar we are using, the use of the PTM acoustic model using the value of 700 for the rec.Pruning and the default value of 1,000,000 for rec.SkipObsFrames results in the best time versus accuracy performance.

VII. CONCLUSION

This work researched whether COTS voice recognition and natural language processing technology could be used to control the interface of a virtual environment, such as NPSNET. While some SLU systems exist in current VE application, such as LeatherNet, the tools used are not available to the public as a COTS product. Also, these systems are not used with the variety of input devices nor do they do as much interactive wargaming as NPSNET. NPSNET provided a challenging environment for a voice system. Overall, the results show that the Nuance Voice Recognition System can be used to build an accurate and responsive voice interface system.

A. ACCOMPLISHMENTS

A SLU grammar was developed to generate the 108 keyboard commands available on the NPSNET keyboard. This was done using Nuance's Grammar Specification Language (GSL) which condensed the writing of the context-free type language. Although recursion is not allowed in Nuance, this did not present a problem when applying the grammar for control purposes. Using the included Nuance tools, such as the random sentence generator and batch processor, the grammar was refined to take into account confusable words and the effects of adding too much flexibility into the grammar by the inclusion of optional words.

The PTM acoustic model was found to be the best speed versus accuracy performer among the included acoustic models in Nuance. The default parameter values and a rec.Pruning value of 700 worked well. Average recognition time was 2.36 seconds with an 83.8% sentence understanding rate. This rate is higher than the exact sentence recognition rate. Therefore, the use of NLP increased the chances of the system understanding the voice input. Importantly, only three percent of the utterances resulted in actions that were not in the same functional category as the action requested.

The SLU system is based on four C++ classes that encapsulate the use of the Nuance API. With these classes, the configuration, voice recognition, and NLP functions

of the system can be used with minimal programming effort. Hence, future work can build on these classes to build voice interfaces into current and future applications. These classes were designed to allow inheritance and polymorphism. They should also be portable to other platforms which run the Nuance system, although this could not be tested since the Nuance system was only available on one SGI system for our development.

A GUI control panel was developed using the SGI RapidApp Application Builder. This panel gives the user feedback as to what the SLU system is doing (listening, processing, ready for speech, etc.), what is the recognition result, and what meaning has been assigned. It also allows the user to look at previous commands and see what the NL interpretation would be. The GUI has been designed to develop into an IDE for voice applications. While not complete, it has a simple editor which allows viewing the currently loaded grammar file, or looking at any other text file in the system.

The SLU system is network capable. From the GUI, a network connection can be opened which uses NPSNET's IDU network and predefined packet structures. It uses multicast addresses to send its data. Therefore it can serve one or more voice-capable NPSNET systems. Minimal changes were made to NPSNET in order to have an internal voice module object listen to the network in the same simulation loop that NPSNET uses to listen to DIS packets which provide necessary information. Once it receives the packet, the voice module translates it into the appropriate keyboard sequence and puts that into the NPSNET keyboard manager. The keyboard manager then processes it as normal input.

B. LESSONS LEARNED

There were several lessons learned in the areas of applied voice recognition, natural language processing, and application development. They were:

- At least one COTS voice recognition/NLP system, the Nuance system, could reliably operate with the large amount of background noise present in the NPSNET graphics laboratory.
- Even in a noisy environment, the lower accuracy acoustic model, PTM, provided about the same recognition and NLP results as the more accurate, but much slower, Genome model. In developing voice application, it is

recommended that the faster, but possibly less accurate, PTM acoustic model be used. A developer should switch to the Genome model if test results with the PTM model are not satisfactory.

- Identifying confusable words is not an intuitive process. There were certain words, such as “double” and “lower”, that the Nuance system consistently got wrong when used in a specific sentence. Surprisingly, this error was not repeated with other sentences legal within the grammar.
- The best way to find sources of error in a grammar is to generate random sentences using the Nuance tools available, record each sentence in a sound file, and perform recognition as a batch process.
- Application generation systems, such as RapidApp, ease the creation of attractive, user-friendly GUI’s. However, they have a large learning curve, especially if they have an underlying object-oriented framework as RapidApp.

C. FUTURE WORK

There is still much that can be done with this project. Currently, this system mimics individual keyboard commands. It has no capability to do two or more operations simultaneously, nor to repetitively issue commands until told to stop, such as to continue turning in a certain direction. Nor does it handle some difficult natural language issues.

Therefore, areas where this topic could be expanded are:

- Add the ability to do more than one action at a time to the system. This would require increasing the grammar while maintaining processing speed and accuracy. It would also require checking to see if the actions are compatible (such as being told to turn left and right), or if they are currently executable.
- Create voice interfaces for the current applications available in NPSNET, this include the OOD, submarine, and Damage Control trainers.
- Develop a grammar and method to query the NPSNET system for information. This is very ambitious. For example, a query about what tanks are near a hill would require getting terrain and entity information and being able to filter it for the results you want.
- Add the ability to handle ellipsis (fragmentary sentences).
- Add the ability to handle anaphora (pronoun resolution).
- Integrate the use of voice with other input devices, such as the cyberglove.

The development of C++ classes and GUI panel provide a foundation for further work in the field of voice interfaces with virtual environments. With it, the task of

implementing a simple interface becomes more a task of developing an effective grammar rather than a programming chore.

The ability of the Nuance system to handle the NPSNET virtual environment indicates that voice recognition systems that use NLP are technologically ready for use in real life military operations. There are several military functions that involve the use of small but well-defined grammars and vocabularies. These include:

- Artillery and naval gunfire Calls-For-Fire (CFF).
- Close Air Support (CAS) nine-line briefs.
- Shiphandling commands.
- Artillery and tank crew commands.

The military can leverage the current state of voice recognition/NLP technology to create more realistic training environments where the user can interact with the system as he would with another human, i.e., by the use of voice rather than a computer mouse or keyboard. The user could interface with complicated Command-and-Control systems which require a large amount of initial and follow-on training. Voice Interfaces with NLP might reduce the training time required to learn and operated such systems. As this thesis illustrates, voice recognition and NLP help a user more fully use systems with a high degree of functionality.

APPENDIX A: NPSNET INTERFACE GRAMMAR

This appendix contains both the initial and updated grammar for this SLU system. The grammars are written in the Nuance System's Grammar Specification Language (GSL). Also included with each grammar is a ramdom-generated list of sentences done by the "random-generate" command included with the Nuance system. By looking at these sentences, one can see if a grammar is well-formed for the intended application and if it is too flexible.

Initial Grammar

```
.NPSNET-Commands
        [   Display-Options             {<category display-options>}
            Entity-Movements            {<category entity-movements>}
            Environment-Effects         {<category environment-effects>}
            Human-Movements             {<category human-movements>}
            Input                       {<category input>}
            Level-Of-Detail             {<category level-of-detail>}
            Sim-Operations              {<category sim-operations>}
            View-Options                {<category view-operations>}
            Weapons                     {<category weapons>}
            DVW                         {<category dvw>}
        ]


Display-Options
        [   Display-Cycle-Cmd           {<action cycle>}
            Display-Toggle-Cmds         {<action toggle>}
            Display-Inc-Dec-Cmds:act    {<action $act>}
        ]
```

```
Display-preamble   ( ?through ?the ?HUD )

HUD        [ (h u d) hud
           ]

NPSNET     [ (n p s) (n p s net) (?the [program simulation])
           ]

Radar-preamble     (?the ?HUD radar)

Display-Cycle-Cmds
           ( cycle Display-preamble
               [   (?text colors)              {<object text-colors>}
                   [   (transparency ?amount)
                       (amount ?(of transparency))
                   ]                           {<object transparency-amount>}
                   (?performer statistics)     {<object performer-statistics>}
                   (?display options)          {<object display-options>}
               ] )


Display-Toggle-Cmds
           (   [toggle (turn [on off])]
               [   (Radar-preamble
                       [   rotation            {<object radar-rotation>}
                           color               {<object radar-color>}
                           icon                {<object radar-icon>}
                       ]   )
                   (?the   [   (display ?(of control measures))
                               (control measures ?display)
                           ]                   {<object control-measure-display>})
                   (?the ?Vehicle ?self ?bounding box)
                                               {<object bounding-box>}
                   (?the   [weapons
                                               {<object weapons-block>}
                           (position ?and ?status)
                                               {<object pos-status-block>}
                           (status ?and ?position)
                                               {<object pos-status-block>}
                       ]  ?block )
                   (?the   [ texturing         {<object texturing>}
                             wireframe         {<object wireframe>}
                           ]
```

```
                    [on                 {<status on>}
                     off                {<status off>}
                    ]   )
            ]   )

Display-Inc-Dec-Cmds
        (   [ increase                  {return (increase)}
              decrease                  {return (decrease)}
            ]
            [   ( ?the ?[(h u d) hud] radar range)
                ( ?the range of ?the ?[(h u d) hud] radar)
            ]                           {<object radar-range>} )

Vehicle     [   vehicle helicopter helo tank (m 1) hind
                airplane plane aircraft
            ]

Speed-Up-Cmds
            [     accelerate (speed up) (raise the speed ?of)
            ]

Slow-Down-Cmds
            [ decelerate (slow down) (lower the speed ?of)
            ]

Entity-Movements
            [   Drive-Cmds
                Pitch-Cmds
                Altitude-Cmds
                Reset-Direction
            ]

Drive-Cmds
            [   (Speed-Up-Cmds ?the ?Vehicle)    {<action speed-up>}
                (Slow-Down-Cmds ?the ?Vehicle)  {<action slow-down>}
                (stop ?the ?Vehicle){<action stop>}
                (   [ roll bank turn]           {<action turn>}
                    [   left                    {<direction left>}
                        right                   {<direction right>}
                    ]   )
            ]                               {<object vehicle>}
```

```
Pitch-Cmds
        (   [   [   increase
                    (pick up)
                     raise
                ]                       {<action increase>}
                [   decrease
                    lower
                ]                       {<action decrease>}
            ]
            ?[the your] [pitch nose]    {<object pitch>} )

Altitude-Cmds
        [   (   [   increase
                    (pick up)
                ]   ?[the your] [altitude elevation] )
            (   [   decrease
                    lower
                ]   ?[the your] [altitude elevation] )
        ]

Reset-Direction
        (reset ?the ?Vehicle direction)     {<action reset>
                                              <object vehicle>
                                              <direction view>}

Environment-Effects
        [   Cloud-Cmds
            Fog-Cmds
        ]

Cloud-Cmds
        [   (   [toggle (turn ?[on off])
                ]                       {<action toggle>}
            ?the clouds ?[on off])      {<object clouds>}
            (   [increase raise double] {<action increase>}
                ?the cloud [ level      {<object cloud-level>}
                             thickness  {<object cloud-thickness>}
                           ] )
            (   [decrease lower]        {<action decrease>}
                ?the cloud [ level      {<object cloud-level>}
                             thickness  {<object cloud-thickness>}
                           ] )
        ]
```

```
Fog-Cmds
        [   (   [toggle (turn ?[on off])
                ]                           {<action toggle>}
            ?the  fog  ?[on off]  )         {<object fog>}
            (   [increase raise]            {<action increase>}
                ?the fog ? [level
                           thickness]  )    {<object fog-thickness>}
            (   [decrease lower]            {<action decrease>}
                ?the  fog ?[level
                           thickness]  )    {<object fog-thickness>}
        ]

Human-Movements
        [   Person-Move-Cmds            {<object soldier>
                                         <action move>}
            Rifle-Cmds                  {<object rifle>
                                         <action aim>}
            Formation-Cmds              {<object soldiers>
                                         <action signal>}
        ]

Person-Move-Cmds
        [   [   (move ?to ?the upright ?position)
                ([stand get] up)
            ]                           {<direction up>}
            (   [move go]
                [   [forward up]        {<direction forward>}
                    down                {<direction down>}
                    left                {<direction left>}
                    right               {<direction right>}
                ]   )
        ]

Rifle-Cmds
        [   (   [point aim sight]       {<action aim> <object sight>}
                *[the toward at]   *[rifle sight]
                [   left                {<direction left>}
                    right               {<direction right>}
                    up                  {<direction up>}
                    down                {<direction down>}
                ]   )
            (   reset                   {<action reset>}
                ?[the my your]
```

```
                [sights (aim point)] )          {<object sights>}
        ]

Formation-Cmds
        (   ?[form (get into)]  ?a
            [   (   echelon                     {<formation echelon>}
                    [   left                    {<direction left>}
                        right                   {<direction right>}
                    ] )
                column                          {<formation column>}
                vee                             {<formation vee>}
                wedge                           {<formation wedge>}
                (open up)                       {<formation open>}
                (close up)                      {<formation close>}
            ]  ?the ?formation )

Input       [   (   reset                       {<action reset>}
                    ?[the my your]  ?resting  [joystick stick controls]
                    ?position)                  {<object resting-stick-position>}
                (   change                      {<action change>}
                    ?[the my your]  ?current input
                    [device method] )           {<object input-device>}
        ]

Level-Of-Detail
        (   [   increase                        {<action increase>}
                [decrease lower]                {<action decrease>}
            ]  ?the
            [   (filter range)
                ( [(l o d) (level of detail)] )
            ] )                                 {<object filter-range>}

Sim-Operations
        [   (   reset                           {<action reset>}
                *[the terrain and static]
                databases)                      {<object databases>}
            (   [   toggle
                    (make [me it (?the Vehicle)] )
                ]                               {<action toggle>}
                [   vulnerable invulnerable
                    vulnerability invulnerability
                ] )                             {<object vulnerability>}
            (   [   [exit quit (get ?me out of)] {<action exit>}
```

```
                    pause                   {<action pause>}
                    resume                  {<action resume>}
                ]
                ?NPSNET)                    {<object npsnet>}
            (   save                        {<action save>}
                ?the ?NPSNET
                [screen image] )            {<object image>}
; these two resets will cause problems with ambiguity if I'm not
; careful. Set up templates for them.
            (   reset                       {<action reset>}
                ?[my the] ?Vehicle
                [location position] )       {<object position>}
            (   [   (reset [me (?my ?the Vehicle)] ?as)
                    (make [me (?my ?the Vehicle)])
                    (bring [me (?my ?the Vehicle)] back)
                ]                           {<action reset>}
                *[up living alive] )        {<object vehicle-state>}
        ]

Weapons [   (   [   toggle
                    (turn [on off])
                ]                           {<action toggle>}
                targeting                   {<object targeting>}
                *[on off] )
            (   cycle                       {<action cycle>}
                ?through ?the
                [munitions weapons]         {<object munitions>}
                ?field )
            (   [fire launch]               {<action fire> <object primary-weapon>}
                ?the ?  [   primary         {<object primary-weapons>}
                            secondary       {<object secondary-weapon>}
                            tertiary        {<object tertiary-weapon>}
                        ]
                ?[weapon guns missile] )
        ]
```

Random Generated List of Sentences from the Initial Grammar

1 top level grammars in npsnetCmds
0: .NPSNET-Commands
Generating from .NPSNET-Commands
0: get out of the simulation
1: quit n p s
2: form a column formation
3: accelerate the airplane
4: get into a open up the formation
5: resume n p s
6: double the cloud thickness
7: save the simulation screen
8: pause
9: exit
10: save the screen
11: slow down vehicle
12: resume the program
13: aim at down
14: get out of the program
15: increase cloud thickness
16: accelerate the helo
17: stand up
18: form a wedge the formation
19: decelerate the
20: go left
21: exit n p s
22: pick up your altitude
23: raise the cloud level
24: move down
25: lower the altitude
26: pause the simulation
27: turn off targeting
28: change my current input method
29: resume the program
30: accelerate the
31: lower the speed of the helo
32: stop the tank
33: turn on the clouds
34: raise the speed of the tank
35: increase the range of the radar
36: exit the program
37: launch the secondary missile
38: resume the simulation
39: exit
40: accelerate tank
41: exit program
42: fire the secondary guns
43: exit the simulation
44: get me out of n p s
45: roll left
46: aim toward the the up
47: turn on display of control measures
48: decrease the range of the hud radar
49: get into a wedge the formation
50: move to upright position
51: vee the
52: double the cloud thickness
53: change my input device
54: get me out of the program
55: lower level of detail
56: launch the secondary weapon
57: quit the program
58: bring me back
59: increase nose
60: speed up the
61: accelerate the tank
62: save the the simulation screen
63: quit simulation
64: launch the primary
65: decrease your pitch
66: point left
67: bank left
68: move to the upright position
69: raise the speed of the m 1
70: save the program image
71: bring helo back living alive
72: resume
73: change the input method
74: decrease the l o d
75: accelerate the tank
76: get out of n p s

77: sight sight sight up
78: go right
79: sight the toward up
80: make it vulnerability
81: accelerate the helicopter
82: decelerate the
83: make it vulnerability
84: exit
85: exit the program
86: pick up the altitude
87: aim toward toward down
88: raise the speed of the
89: slow down the helicopter
90: get into a column the
91: speed up the airplane
92: go right
93: pause the program
94: accelerate the plane
95: resume the program
96: pick up altitude
97: accelerate helicopter
98: aim the toward the at the the rifle sight rifle down
99: exit the program
100: quit n p s
101: double the cloud thickness
102: lower the cloud thickness
103: save the the simulation screen
104: wedge formation
105: go left
106: increase the nose
107: exit the program
108: sight sight sight sight down
109: reset me as
110: pause program
111: make me vulnerable
112: get into a close up the
113: toggle display
114: a column the
115: roll left
116: exit
117: double the cloud level
118: point the rifle rifle right
119: decrease the elevation
120: double the cloud level
121: fire secondary guns
122: change your input method
123: exit the program
124: form a wedge the formation
125: raise the speed of the airplane
126: get out of
127: lower the cloud level
128: pick up the altitude
129: save the program screen
130: get into column the formation
131: roll left
132: exit
133: launch tertiary
134: roll right
135: go up
136: pause program
137: get into a close up formation
138: fire tertiary missile
139: get out of the program
140: go down
141: launch the primary
142: speed up the tank
143: get up
144: stand up
145: pause
146: decrease the l o d
147: increase the fog
148: go forward
149: roll left
150: resume the simulation
151: point toward the up
152: quit n p s net
153: turn on targeting
154: save the n p s image
155: pick up your altitude
156: get into a wedge the formation
157: a echelon left the
158: exit the simulation
159: lower the pitch
160: resume
161: get out of n p s net

162: point at rifle sight rifle up
163: slow down the helicopter
164: change the current input method
165: pause
166: decrease the elevation
167: pause program
168: speed up the aircraft
169: go forward
170: point the the rifle sight rifle down
171: quit the program
172: make me vulnerability
173: roll left
174: form close up
175: speed up the plane
176: slow down helo
177: go down
178: pick up altitude
179: bank right
180: resume the program
181: launch weapon
182: stop airplane
183: resume the simulation
184: quit
185: make the hind vulnerability
186: stop the
187: toggle the clouds on
188: aim toward the right
189: go left
190: speed up the helicopter
191: decelerate the helicopter
192: increase fog level
193: stand up
194: change the input device
195: lower the speed of the vehicle
196: resume the simulation
197: roll left
198: resume the program
199: pause n p s

Updated Grammar

```
.NPSNET-Commands
        [   Display-Options             {<category display-options>}
            Entity-Movements            {<category entity-movements>}
            Environment-Effects         {<category environment-effects>}
            Human-Movements             {<category human-movements>}
             Input                      {<category input>}
            Level-Of-Detail             {<category level-of-detail>}
            Sim-Operations              {<category sim-operations>}
            View-Options                {<category view-operations>}
            Weapons                     {<category weapons>}
            DVW                         {<category dvw>}
        ]


Display-Options
        [   Display-Cycle-Cmd           {<action cycle>}
            Display-Toggle-Cmds         {<action toggle>}
            Display-Inc-Dec-Cmds:act    {<action $act>}
        ]

Display-preamble   ( ?through the ?HUD )
HUD     [ (h u d) hud
        ]

NPSNET  [ (n p s) (n p s net) (the [program simulation])
        ]

Radar-preamble     (?HUD radar)

Display-Cycle-Cmds
        (  [cycle change (turn [on off]) ]
            [ ( Display-preamble
            [   (text colors)           {<object text-colors>}
                [   (transparency ?amount)
                    (amount ?(of transparency))
                ]                       {<object trans-amount>}
                (?display options)      {<object display-opts>}
            ] )
            (?performer statistics)     {<object pf-stats>}
        ] )
```

```
Display-Toggle-Cmds
        (   [toggle (turn [on off])]  the
            [   (Radar-preamble
                    [   rotation            {<object radar-rotation>}
                        color               {<object radar-color>}
                        icon                {<object radar-icon>}
                    ]   )
                (   [   (display of control measures)
                        (control measures ?display)
                    ]                       {<object controls>})
                (?self ?bounding box)       {<object bounding-box>}
                (   [weapons                {<object weapons-block>}
                        (position ?and ?status)
                                            {<object pos-stat-block>}
                        (status ?and ?position)
                                            {<object pos-stat-block>}
                    ]
                    [block information]  )
                (   [ texturing             {<object texturing>}
                      wireframe             {<object wireframe>}
                    ]
                )
            ]   )

Display-Inc-Dec-Cmds
         (  [ increase                      {return (increase)}
              decrease                      {return (decrease)}
            ]
            [   ( the ?HUD radar range)
                ( the range of the ?HUD radar)
            ]                               {<object radar-range>} )

Vehicle     [   vehicle helicopter helo tank (m 1) hind
                airplane plane aircraft
            ]

Speed-Up-Cmds
            [       (increase speed) accelerate (speed up)
                    (raise the speed ?of)
            ]

Slow-Down-Cmds
            [       (slow down) (lower the speed ?of) ]
```

```
Entity-Movements
        [   Drive-Cmds
            Pitch-Cmds
            Altitude-Cmds
            Reset-Direction
        ]

Drive-Cmds
        [   (Speed-Up-Cmds ?the ?Vehicle)     {<action speed-up>}
            (Slow-Down-Cmds ?the ?Vehicle)  {<action slow-down>}
            (stop ?the ?Vehicle){<action stop>}
            (   [ roll bank turn]           {<action turn>}
                [   left                    {<direction left>}
                    right                   {<direction right>}
                ]   )
        ]                                   {<object vehicle>}

Pitch-Cmds
        (   [   [   increase (pick up) raise ] {<action increase>}
                [   decrease lower ]        {<action decrease>}
            ]
            ?[the your] [pitch nose]        {<object pitch>} )

Altitude-Cmds
        [   (   increase                    {<action increase>}
                ? [the your]
                [altitude elevation]        {<object altitude>}
            )
            (   [ decrease lower ]          {<action decrease>}
                ?[the your]
                [altitude elevation]        {<object altitude>}
            )
        ]

Reset-Direction
        (reset ?the ?Vehicle direction)     {<action reset>
                                              <object vehicle>
                                              <direction view>}

Environment-Effects
        [   Cloud-Cmds
            Fog-Cmds
        ]
```

```
Cloud-Cmds
        [   (   [toggle (turn [on off]) ]   {<action toggle>}
                ?the clouds)                {<object clouds>}
            (   [increase raise double]     {<action increase>}
                ?the cloud [ level          {<object cloud-level>}
                             thickness      {<object cloud-thick>}
                           ] )
            (   [decrease lower]            {<action decrease>}
                ?the cloud [ level          {<object cloud-level>}
                             thickness      {<object cloud-thick>}
                           ] )
        ]

Fog-Cmds
        [   (   [toggle (turn [on off] ) ]  {<action toggle>}
                the  fog)                   {<object fog>}
            (   [increase raise]            {<action increase>}
                the fog ? [level
                           thickness]  )    {<object fog-thick>}
            (   [decrease lower]            {<action decrease>}
                the  fog ?[level
                           thickness]  )    {<object fog-thick>}
        ]

Human-Movements
        [   Person-Move-Cmds                {<object soldier>
                                             <action move>}
            Rifle-Cmds                      {<object rifle>
                                             <action aim>}
            Formation-Cmds                  {<object soldiers>
                                             <action signal>}
        ]

Person-Move-Cmds
        [   (   [stand get]  up)            {<direction up>}
            (   [move go]
                [   [forward up]            {<direction forward>}
                    down                    {<direction down>}
                    left                    {<direction left>}
                    right                   {<direction right>}
                ]   )
            (get down)                      {<direction down>}
        ]
```

```
Rifle-Cmds
        [   (   [   (   [point aim] ?the
                        ?[rifle sight] ?[toward at]  ?the)
                    (   (sight in) [to toward on] ?the)
                ]
                [   left                    {<direction left>}
                    right                   {<direction right>}
                    up                      {<direction up>}
                    down                    {<direction down>}
                ]   )
            (   reset                       {<action reset>}
                ?[the my your]
                [sights (aim point) ] )     {<object sights>}
        ]

Formation-Cmds
        [   (   [form (get into)]  a
                [   (   echelon             {<formation echelon>}
                        [ left              {<direction left>}
                          right             {<direction right>}
                        ] )
                    column                  {<formation column>}
                    vee                     {<formation vee>}
                    wedge                   {<formation wedge>}
                ]  ?the ?formation
            )
            (   [   (open up)               {<formation open>}
                    (close up)              {<formation close>}
                ] the formation
            )
        ]


Input   [   (   reset                       {<action reset>}
                ?[the my your]  ?resting  [joystick stick controls]
                ?position)                  {<object stick>}
            (   change                      {<action change>}
                ?[the my your]  ?current input
                [device method] )           {<object input-device>}
        ]
```

```
Level-Of-Detail
            (   [   increase                    {<action increase>}
                    [decrease lower]            {<action decrease>}
                ]   the
                [   (filter range)
                    ( [(l o d) (level of detail)] )
                ] )                             {<object filter-range>}

Sim-Operations
            [   (   reset                       {<action reset>}
                    (   (the terrain) ? (and static) )
                    databases)                  {<object databases>}
                (   [   toggle
                        (make [me it (?the Vehicle)] )
                    ]                           {<action toggle>}
                    [   vulnerable invulnerable
                        vulnerability invulnerability
                    ] )                         {<object vulnerability>}
                (   [   [exit quit (get ?me out of)] {<action exit>}
                        pause                   {<action pause>}
                        resume                  {<action resume>}
                    ]
                    ?NPSNET)                    {<object npsnet>}
                (   save                        {<action save>}
                    ?the ?NPSNET
                    [screen image] )            {<object image>}
; these two resets will cause problems with ambiguity if I'm not
; careful. Set up templates for them.
                (   reset                       {<action reset>}
                    ?[my the] ?Vehicle
                    [location position] )       {<object position>}
                (   [   (reset [me (?my ?the Vehicle)] ?as)
                        (make [me (?my ?the Vehicle)])
                        (bring [me (?my ?the Vehicle)] back)
                    ]                           {<action reset>}
                    *[up living alive] )        {<object vehicle-state>}
            ]

Weapons     [   (   [   toggle
                        (turn [on off])
                    ]                           {<action toggle>}
                    targeting                   {<object targeting>}
                )
```

```
    (   cycle                       {<action cycle>}
        ?through ?the
        [munitions weapons]         {<object munitions>}
        ?field )
    (   [fire launch]               {<action fire> <object primary>}
        ?the ?  [   primary         {<object primary>}
                    secondary       {<object secondary>}
                    tertiary        {<object tertiary>}
                ]
        [weapon guns missile] )
]
```

List of Random Generated Sentences from the Updated Grammar

1 top level grammars in npsnetCmds1
0: .NPSNET-Commands
Generating from .NPSNET-Commands
0: exit the simulation
1: roll left
2: roll left
3: sight in on the right
4: aim the at the right
5: decrease your nose
6: point the toward the down
7: accelerate the m 1
8: get down
9: form a column the
10: quit n p s net
11: save the image
12: double the cloud level
13: turn on targeting
14: roll left
15: move up
16: aim the rifle left
17: aim right
18: slow down the
19: accelerate the m 1
20: save the screen
21: pick up the pitch
22: change the h u d display
23: change the statistics
24: slow down the
25: bring my tank back alive
26: resume the simulation
27: toggle targeting
28: change my input method
29: roll right
30: turn on the fog
31: decrease the cloud level
32: resume n p s net
33: get into a wedge the formation
34: pause the program
35: point the toward the left
36: quit the program
37: resume n p s
38: change the hud transparency
39: point the up
40: aim the rifle toward the right
41: double cloud level
42: go down
43: stop the helicopter
44: move forward
45: get into a wedge the
46: lower the elevation
47: get into a vee the formation
48: bank right
49: close the formation
50: get down
51: go right
52: pick up nose
53: move down
54: raise the pitch
55: form a vee the formation
56: quit the program
57: resume the program
58: resume the simulation
59: toggle the control measures display
60: slow down the
61: fire the tertiary guns
62: close up the formation
63: save the the simulation screen
64: fire the tertiary weapon
65: roll left
66: decrease the fog thickness
67: open up the formation
68: turn on the fog
69: save the the simulation image
70: double the cloud thickness
71: form a column the
72: exit the program
73: turn off the fog
74: fire the primary guns
75: turn on the clouds
76: save the image
77: change the hud transparency amount
78: turn off the control measures
79: exit the simulation
80: increase the h u d radar range

81: roll left
82: double the cloud level
83: increase the fog thickness
84: cycle through the munitions field
85: make the vehicle vulnerable
86: pause n p s net
87: slow down the vehicle
88: aim sight at left
89: aim the rifle at up
90: roll right
91: reset the helicopter
92: save the the simulation image
93: slow down the plane
94: fire secondary missile
95: pause the program
96: decrease the cloud thickness
97: get down
98: stand up
99: increase the fog
100: go down
101: double the cloud thickness
102: decrease the l o d
103: change the hud transparency
104: toggle targeting
105: open up the formation
106: launch secondary weapon
107: exit the program
108: move left
109: toggle targeting
110: toggle invulnerability
111: stop aircraft
112: change the performer statistics
113: move left
114: change through the hud display options
115: go left
116: bank right
117: get out of
118: get up
119: double the cloud level
120: close up the formation
121: launch the secondary missile
122: save the program image
123: resume the program
124: get me out of
125: form a echelon left the
126: close the formation
127: reset the terrain and static databases
128: aim the sight toward the right
129: change the current input method
130: lower the cloud level
131: turn on targeting
132: turn right
133: pause n p s net
134: exit n p s
135: raise the speed of the
136: exit n p s
137: quit the simulation
138: go down
139: change the input device
140: get me out of n p s
141: bank right
142: make hind invulnerable
143: resume the simulation
144: cycle the munitions
145: save the the program screen
146: close the formation
147: resume n p s net
148: accelerate the tank
149: turn off the fog
150: get up
151: point the rifle toward right
152: double the cloud thickness
153: bank left
154: change the hud display options
155: exit
156: accelerate the helo
157: point the rifle toward the left
158: speed up the helo
159: resume
160: turn on the position status block
161: exit
162: turn on the fog
163: decrease the level of detail
164: open up the formation
165: change input device
166: move right
167: launch the tertiary weapon
168: go forward
169: lower your pitch
170: go left
171: double the cloud level
172: quit the program

173: save the n p s net image
174: speed up plane
175: bring the hind back
176: bank left
177: decrease the nose
178: exit the program
179: increase the range of the h u d radar
180: open the formation
181: reset the terrain and static databases
182: accelerate m 1
183: point the sight toward the right
184: open up the formation
185: decrease the range of the hud radar
186: launch the secondary weapon
187: roll left
188: raise your nose
189: close the formation
190: speed up vehicle
191: get down
192: lower the speed the m 1
193: make me invulnerability
194: pick up the nose
195: double the cloud thickness
196: reset my the tank
197: resume the simulation
198: fire the weapon
199: bring my the hind back

APPENDIX B: SOURCE CODE

Appendix B contains the source code for non-RapidApp generated files. These include the files for the following C++ classes:

- configClass
- recClientClass
- NLClass
- recognizerClass
- voiceVkApp
- voiceNetManagerClass
- voiceModuleClass

It reproduces the following files:

All the software code in this thesis is subject to the following copyright notice:

Graphics and Video Laboratory and the NPSNET Research Group be given written credit in your software's written documentation and be given graphical credit on any start-up/ credit screen your software generates. This restriction helps justify our research efforts to the sponsors who fund our research.

Do not redistribute this code without the express written consent of the NPSNET Research Group. (E-mail communication and our confirmation qualifies as written permission.) As stated above, this restriction helps justify our research efforts to the sponsors who fund our research.

This software was designed and implemented at U.S. Government expense and by employees of the U.S. Government. It is illegal to charge any U.S. Government agency for its partial or full use.

THE SOFTWARE IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

E-Mail addresses:
npsnet@cs.nps.navy.mil
General code questions, concerns, comments, requests for distributions and documentation, and bug reports.
npsnet-info@cs.nps.navy.mil
Contact principle investigators.
Overall research project information and funding.
Requests for demonstations.

```
//*****************************************************************************
//
//   File: errorReportingClass.h
//   Purpose: Gives the class definition of an error reporting class. This
//            gives the error report functions that are common throughout the
//            applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 10 July 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//
//*****************************************************************************
#ifndef __ERRORREPORTINGCLASS_H
#define __ERRORREPORTINGCLASS_H

#include "corona-config.h"

class errorReportingClass {

public:
   errorReportingClass(){};
   ~errorReportingClass(){};

protected:
   void check(CoronaStatus, int) const;
   void usage(const char *) const;
   void localFatalError(char *routine, char *format, void *arg1,
                        void *arg2, void *arg3) const;
   void localWarning(char *routine, char *format, void *arg1,
                     void *arg2, void *arg3) const;

};

#endif

//End-Of-File errorReportingClass.h
```

```
//*****************************************************************************
//   File: errorReportingClass.C
//   Purpose: Gives the class definition of an error reporting class. This
//            gives the error report functions that are common throughout the
//            applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 20 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//*****************************************************************************
#include <stdlib.h>
#include <iostream.h>
#include "errorReportingClass.h"




//*****************************************************************************
//     Function: usage
//     Purpose:  Tells the user how he should call the function
//     Parameters: Char * - text string with info
//     Return:   void
//     Last Date Modified: 20 Aug 1996
//     Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//*****************************************************************************
void errorReportingClass::usage(const char *prog) const
{
    cerr << "\nUSAGE: " << prog << " -package <recognition-package> "
         << "[Corona Toolkit Options]\n" << endl;
    exit(0);
} //End usage


//*****************************************************************************
//     Function: localFatalError
//     Purpose:  Gives error information in event of a bad status return
//     Parameters: char *routine - name of routine where the error occurred.
//                 char *format - Gives format for next 3 string parameters
//                 void *arg1, *arg2, *arg3 - optional data to be printed out
//                    in the specified format.
//     Return:   void
//     Last Date Modified: 20 Aug 1996
//     Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//*****************************************************************************
void errorReportingClass::localFatalError(char *routine, char *format,
                                  void *arg1, void *arg2, void *arg3) const
{
    cerr << "   ****** F A T A L - E R R O R ******" << endl;
    fprintf(stderr, "%s: ", routine);
    fprintf(stderr, format, arg1, arg2, arg3);
    exit(-11);
} //End localFatalError
```

```
//***************************************************************************
//     Function: localWarning
//     Purpose:  Gives warning information
//     Parameters: char *routine - name of routine where the error occurred.
//                 char *format - Gives format for next 3 string parameters
//                 void *arg1, *arg2, *arg3 - optional data to be printed out
//                    in the specified format.
//     Return:   void
//     Last Date Modified: 10 July 1996
//     Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
void errorReportingClass::localWarning(char *routine,char *format, void *arg1,
                                       void *arg2, void *arg3) const
{
    cerr << "Warning: " << routine << endl;
    fprintf(stderr, format, arg1, arg2, arg3);

    return;
} //End localWarning


//***************************************************************************
//     Function: check
//     Purpose:  Check the status return when executing Nuance API's.
//               If a bad status is returned, then it exits the program
//               showing error information.
//     Parameters: CoronaStatus - Contains the status to check
//                 int - Holds what source code line number the error
//                       occurred.
//     Return:   void
//     Last Date Modified: 10 July 1996
//     Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
void errorReportingClass::check(CoronaStatus stat, int line) const
{
    if (CORONA_OK != stat){
       localFatalError("",
                       "Function failed at line %d because '%s'\n",
                       (void *) line, CoronaErrorMessage(stat), NULL);
    }
} //End check


//End-Of-File errorReportingClass.C
```

```
//***************************************************************************
//   File: configClass.h
//   Purpose: Gives the class definition of the Nuance config class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 25 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __CONFIGCLASS_H
#define __CONFIGCLASS_H

#include "errorReportingClass.h"


class configClass : protected errorReportingClass {

public:

   //Constructors for the class. Implements the four ways Nuance provides
   //to initialize the recognizer client.
                configClass(int *, char **, int = 1);
                configClass(char *);
                configClass(FILE *, char *, int = 1, int = 1,
                            CoronaConfig * = NULL);
                configClass(int *, char **, char *, int = 1, int = 1,
                            int = 1, CoronaConfig * = NULL);

   //Destructor
                ~configClass();

   //Default copy and operator = constructors
                configClass(const configClass &);
   configClass&   operator=(const configClass &);

   //These functions get internal values of the config object.
   CoronaConfig  *getConfig() const;
   CoronaStatus   getStatus() const;
   char          *getPackageLocation() const;
   char          *getPackageID() const;
   char         **getGrammarNames() const;
   int            getNumberOfGrammars() const;
   int            isNLDefined() const;

   //May need to set the NLDefined private member, esp. from NLClass
   void           setNLDefined(int);

   //Print out all the config values. Useful for debugging.
   void           print() const;

private:
   void           makeGrammarNamesList();
   void           makePackageLocation();
```

```
   void           makePackageID();
   CoronaConfig  *config;
   CoronaStatus   status;
   char         **grammarList;
   int            numberOfGrammars;
   char          *packageLocation;
   char          *packageID;
   int            NLDefined;
};

#endif

//End-Of-File configClass.h
```

```
//***************************************************************************
//
//   File: configClass.C
//   Purpose: Gives the class definition of the Nuance config class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 21 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//
//***************************************************************************
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
#include "configClass.h"


//***************************************************************************
//     Function: Constructor using CoronaBuildFromCommandLine
//     Purpose:  Most flexible way to initialize system using command line
//               arguments coming from the main() routine.
//     Parameters: int * argc_ptr - gives number of commandline args passed in
//                 char **argv - actual argument strings
//                 int package_required - Set to zero if not using the
//                                        recognition abilities of Nuance
//     Return:   void - Status of init. is kept in status member.
//     Last Date Modified: 30 Aug 1996
//***************************************************************************
configClass::configClass(int *argc_ptr, char **argv_ptr, int package_required)
{
    config = CoronaConfigBuildFromCommandLine(argc_ptr, argv_ptr,
package_required,
                                          &status);

    if (!config){
       usage(argv_ptr[0]);
       localWarning("", "%s: Failed to initialize configuration because '%s'\n",
                             argv_ptr[0], CoronaErrorMessage(status), NULL);
    }

    int arg_upto = 1;
    while (arg_upto < *argc_ptr) {
             if (!strcmp(argv_ptr[arg_upto], "-norej")) {
             }
        else {
            cerr << "ERROR: unknown argument: \""
                           << argv_ptr[arg_upto] << endl;
            usage(argv_ptr[0]);
        }
        arg_upto++;
    }

    //Make the package specific information
```

```
    makePackageLocation();
    makePackageID();
    makeGrammarNamesList();

    //Is NL defined for this package?
    status = CoronaConfigGetIntParameter(config, "package.NLDefined",
                                         &NLDefined);
    check(status, __LINE__);

    cout << "ConfigClass Object has been instantiated" << endl << endl;
    return;
} //End configClass



//*************************************************************************
//     Function: Constructor using CoronaConfigBuild
//     Purpose:  Only asks for a directory name.
//     Parameters: char * - Name of the directory the compiled package is in.
//     Return:   none
//     Last Date Modified: 30 Aug 1996
//*************************************************************************
configClass::configClass(char * packageDir)
{
    config = CoronaConfigBuild(packageDir, &status);

    if (!config){
       localFatalError("",
                       "%s: Failed to initialize configuration because '%s'\n",
                                "CoronaConfigBuild",
CoronaErrorMessage(status), NULL);
    }

    //Make the package specific information
    makePackageLocation();
    makePackageID();
    makeGrammarNamesList();

    //Is NL defined for this package?
    status = CoronaConfigGetIntParameter(config, "package.NLDefined",
                                         &NLDefined);
    check(status, __LINE__);

    cout << "ConfigClass Object has been instantiated" << endl << endl;
    return;
} //End configClass
```

```
//*************************************************************************
//     Function: Constructor using CoronaConfigFromFile
//     Purpose:  This can create a new config object, or it can overwrite or
//               add settings to an existing config object.
//     Parameters: FILE * fileFD - file id that holds config parameters
//                 char *paramSource -
//                 int fromUser - Only user settable values can be entered
//                 int mustBeValid - Causes validity checks of parameter values
//                 CoronaConfig * config - if NULL, creates a new object
//     Return:   none
//     Last Date Modified: 30 Aug 1996
//*************************************************************************
configClass::configClass(FILE * fileFD, char *paramSource, int fromUser,
                         int mustBeValid, CoronaConfig * Config)
{
    config = CoronaConfigFromFile(Config, fileFD, paramSource, fromUser,
                                  mustBeValid, &status);

    if (!config){
       localFatalError("",
                       "%s: Failed to initialize configuration because '%s'\n",
                                  "CoronaConfigFromFile",
CoronaErrorMessage(status), NULL);
    }

    //Make the package specific information
    makePackageLocation();
    makePackageID();
    makeGrammarNamesList();

    //Is NL defined for this package?
    status = CoronaConfigGetIntParameter(config, "package.NLDefined",
                                         &NLDefined);
    check(status, __LINE__);

    cout << "ConfigClass Object has been instantiated" << endl << endl;
    return;
} //End configClass
```

```
//*************************************************************************
//     Function: Constructor using CoronaConfigFromStringArray
//     Purpose:  This constructor can make a configClass object using param's
//               passed in from a string array.The string array can be modified.
//               It also can override current parameter values.
//     Parameters:
//     Return:   none
//     Last Date Modified: 30 Aug 1996
//*************************************************************************
configClass::configClass(int *stringCount, char **string, char *param_source,
                         int fromUser, int mustBeValid, int removeArgs,
                         CoronaConfig *Config)
{
    config = CoronaConfigFromStringArray(Config, stringCount, string,
                                         param_source, fromUser, mustBeValid,
                                         removeArgs, &status);

    if (!config){
       localFatalError("","%s:Failed to initialize configuration because'%s'\n",
                       "CoronaConfigFromStringArray",
                       CoronaErrorMessage(status), NULL);
    }

    //Make the package specific information
    makePackageLocation();
    makePackageID();
    makeGrammarNamesList();

    //Is NL defined for this package?
    status = CoronaConfigGetIntParameter(config, "package.NLDefined",
                                         &NLDefined);
    check(status, __LINE__);

    cout << "ConfigClass Object has been instantiated" << endl << endl;
    return;
} //End configClass


//*************************************************************************
//     Function: Destructor
//     Purpose:  Uses CoronaConfigFree to deallocate memory
//     Parameters: None
//     Return:   void
//     Last Date Modified: 20 Aug 1996
//*************************************************************************
configClass::~configClass()
{
    CoronaConfigFree(config);

    for (int ix = 0; ix < numberOfGrammars; ix++) {
              delete [] grammarList[ix];
    }

    delete [] grammarList;
    delete [] packageLocation;
```

```
    delete [] packageID;

    cerr << "ConfigClass Destructor is complete" << endl;
} //End Destructor


//***************************************************************************
//     Function: configClass
//     Purpose:  Copy constructor
//     Parameters: const configClass &
//     Return:   none
//     Last Date Modified: 21 Aug 1996
//***************************************************************************
configClass::configClass(const configClass& configObj)
{
    config = configObj.getConfig();
    status = configObj.status;
    grammarList = configObj.getGrammarNames();
    numberOfGrammars = configObj.numberOfGrammars;
    packageLocation = configObj.getPackageLocation();
    packageID = configObj.getPackageID();
    NLDefined = configObj.NLDefined;
} //End copy constructor


//***************************************************************************
//     Function: operator=
//     Purpose:  prevent shallow copy during assignment
//     Parameters: const configClass &
//     Return:   configClass &
//     Last Date Modified: 21 Aug 1996
//***************************************************************************
configClass & configClass::operator=(const configClass& configObj)
{
    config = configObj.config;
    status = configObj.status;

    //To copy grammar list, the old list must be deleted first,
    //then the new one can be created.
    for (int ix = 0; ix < numberOfGrammars; ix++) {
              delete [] grammarList[ix];
    }
    delete [] grammarList;
    grammarList = configObj.getGrammarNames();

    numberOfGrammars = configObj.numberOfGrammars;
    packageLocation = configObj.getPackageLocation();
    packageID = configObj.getPackageID();
    NLDefined = configObj.NLDefined;

    return (*this);
} //End operator =
```

```
//*************************************************************************
//     Function: getConfig
//     Purpose:  Get the pointer to a copy of the Nuance config object.
//               Copy is made so a user cannot destroy the original object.
//               This value is used by both Rec and NL objects for
//               initialization.
//        NOTE:  The new CoronaConfig object created needs to be destroyed
//               by the calling program.
//     Parameters: void
//     Return:   CoronaConfig * - gives pointer to copy of config.
//     Last Date Modified: 8 July 1996
//*************************************************************************
CoronaConfig * configClass::getConfig() const
{
    CoronaStatus stat;
    CoronaConfig *newConfig = CoronaConfigCopy(config, &stat);
    check(stat,  __LINE__);
    return newConfig;
} //End getConfig


//*************************************************************************
//     Function: getStatus
//     Purpose:  returns the status of config object init. or actions
//     Parameters: void
//     Return:   CoronaStatus - gives the status of the config object
//     Last Date Modified: 8 July 1996
//*************************************************************************
CoronaStatus configClass::getStatus() const
{
    return status;
} //End getStatus


//*************************************************************************
//     Function: getPackageLocation
//     Purpose:  Gives the directory where the current package comes from.
//        NOTE:  The char * pointer needs to be deleted by the calling program.
//     Parameters: Void
//     Return:   char* - dynamically allocated string address holding
//        packages source directory.
//     Last Date Modified: 20 Aug 1996
//*************************************************************************
char * configClass::getPackageLocation() const
{
    char *location = strdup(packageLocation);
    return location;
} //End getPackageLocation
```

```
//*************************************************************************
//     Function: getPackageID
//     Purpose:  Gives the package name or "ID".
//        NOTE:  The char * pointer needs to be deleted by the calling program.
//     Parameters: void
//     Return:   char* - dynamically allocated string address holding
//        packages source directory.
//     Last Date Modified: 20 Aug 1996
//*************************************************************************
char * configClass::getPackageID() const
{
    char *ID = strdup(packageID);
    return ID;
} //End getPackageID



//*************************************************************************
//     Function: getGrammarNames
//     Purpose:  Reproduce the internal listing of grammar names held
//               in the current package.
//        NOTE:  The char ** pointer passed needs to be deleted by the
//               calling program.
//     Parameters: void
//     Return:   char ** - a pointer to a list of strings
//     Last Date Modified: 20 Aug 1996
//*************************************************************************
char ** configClass::getGrammarNames() const
{
    char **list = new char*[numberOfGrammars];

    //Do memberwise copy of grammar names
    for (int ix = 0; ix < numberOfGrammars; ix++) {
       list[ix] = strdup(grammarList[ix]);
    }

    return list;
} //End getGrammarNames


//*************************************************************************
//     Function: getNumberOfGrammars
//     Purpose:  Gets the number of top-level grammars present in the package
//     Parameters: void
//     Return:   int - number of top-level grammars
//     Last Date Modified: 8 July 1996
//*************************************************************************
int configClass::getNumberOfGrammars() const
{
    return numberOfGrammars;
} //End getNumberOfGrammars
```

```
//***************************************************************************
//     Function: isNLDefined
//     Purpose:  Queries whether NL is defined in this package
//     Parameters: void
//     Return:   int - true or false about NL use in this package
//     Last Date Modified: 8 July 1996
//***************************************************************************
int configClass::isNLDefined() const
{
    return NLDefined;
}


//***************************************************************************
//     Function: setNLDefined
//     Purpose:  Sets the NLDefined data member
//     Parameters: int - true or false value
//     Return:   void
//     Last Date Modified: 10 July 1996
//***************************************************************************
void configClass::setNLDefined(int setting)
{
    //Check if it is negative. If so, give a warning, and set to true
    if (setting < 0) {
       cout << "The setting for NLDefined is negative." << endl
            << "Assuming that the requested setting is TRUE"
            << endl << endl;
       setting = 1;
    }

    NLDefined = setting;
    return;
} //End setNLDefined


//***************************************************************************
//     Function: print
//     Purpose:  Print out config object values. Good for debugging.
//     Parameters: void
//     Return:   void
//     Last Date Modified: 6 June 1996
//***************************************************************************
void configClass::print() const
{
    CoronaConfigPrint(config);
} //End print
```

```
//*****************************************************************************
//     Function: makePackageLocation
//     Purpose:  Stores the directory name where the package is located
//     Parameters: void
//     Return:   void
//     Last Date Modified: 8 July 1996
//*****************************************************************************
void configClass::makePackageLocation()
{
    //Local Variables
    int length = 0;
    const int BUFFERLEN = 100;
    char buffer[BUFFERLEN];

    //Get the directory location. Check for errors.
    status = CoronaConfigGetStringParameter(config, "package.Location",
                                            buffer, BUFFERLEN);
    check(status, __LINE__);

    //Allocate memory for the string, then store it in the data member.
    length = strlen(buffer);
    packageLocation = new char[length + 1];
    strncpy(packageLocation, buffer, length);

cerr << "Package Location is set to: " << buffer << endl;
    return;
} //End makePackageLocation


//*****************************************************************************
//     Function: makePackageID
//     Purpose:  Utility func that store the string ID of the package.
//     Parameters: void
//     Return:   void
//     Last Date Modified: 6 June 1996
//*****************************************************************************
void configClass::makePackageID()
{
    //Local Variables
    int length = 0;
    const int BUFFERLEN = 100;
    char buffer[BUFFERLEN];

    //Get the ID string. Check for errors.
    status = CoronaConfigGetStringParameter(config, "package.ID", buffer,
                                            BUFFERLEN);
    check(status, __LINE__);

    //Allocate memory for the string, then store it in the data member.
    length = strlen(buffer);
    packageID = new char[length + 1];
    strncpy(packageID, buffer, length);

    return;
} //End makePackageID
```

```
//*************************************************************************
//     Function: makeGrammarNamesList
//     Purpose:  Utility func. that creates a list of top level grammars
//     Parameters: void
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void configClass::makeGrammarNamesList()
{
    //Local variables
    int i;
    char paramName[1000];
    char grammarName[1000];

    //First, find the number of grammars in the package
    status = CoronaConfigGetIntParameter(config, "package.NumGrammars",
                                         &numberOfGrammars);
    check(status, __LINE__);

    //Allocate enough space for the grammar list
    grammarList = new char *[numberOfGrammars];

    //Copy each grammar name into the grammar list
    for (i=0; i < numberOfGrammars; i++) {
        sprintf(paramName, "package.GrammarName%d", i);
        status = CoronaConfigGetStringParameter(config,
                                                paramName,
                                                grammarName,
                                                sizeof(grammarName));
        if (CORONA_OK != status){
           localFatalError("build_grammar_list",
                           "Unable to find param '%s'\n",
                           paramName, NULL, NULL);
        }
     grammarList[i] = (char *)strdup(grammarName);
     }

     return;
} //End makeGrammarNamesList


//End-Of-File configClass.C
```

```
//***************************************************************************
//   File: recClientClass.h
//   Purpose: Gives the class definition of the Nuance RecClient class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 23 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __RECCLIENTCLASS_H
#define __RECCLIENTCLASS_H

#include <Xm/Xm.h>
#include "errorReportingClass.h"
#include "configClass.h"
#include "recclient.h"

class recClientClass : protected errorReportingClass {

public:
    //Constructors, operator =, and destructor
                recClientClass(configClass *, int = 60, XtAppContext = NULL);
                recClientClass(const recClientClass &);
                recClientClass &operator=(const recClientClass &);
               ~recClientClass();

    //Start, stop, set callbacks functions
    virtual void listen(char * = NULL, float = 3.0);
    virtual void startListening(char * = NULL);
    virtual void stopListening();
    virtual void abort();
    virtual void regCallback(CoronaEvent, RCCallbackFnPtr);

    //Utterance playback/stop functions
    virtual void playFile(char *, int = 60);
    virtual void playLastUtterance(int = 60);
    virtual void killPlayback();

    //Query functions and parameter setting
    virtual void getResults(int, void *) const;
    virtual RecClient *getRecClient() const;
    virtual void setParameter(int, int);
    virtual void setParameter(int, char);
    virtual void getParameter(int, int *) const;
    virtual void getParameter(int, char *) const;
    virtual int  isWaitingForEvent() const;

    //Recognizer results, open to the public
    RecResult   *resultsPtr;

protected:
    //Set up X callbacks in C++ environment
    static void  suddenDeathCB();
```

```
    static void  processEventsCB(recClientClass **);
    virtual void processEvents(RecClient *);
    virtual void setupSuddenDeath();

    virtual void initRecClient(int, XtAppContext);
    CoronaConfig *configPtr;
    RecClient   *clientPtr;
    int          recFileDescriptor;
    int          waitingForEvent;
    CoronaStatus status;
    recClientClass * recClientClassObjectPtr;
};

#endif

//End-Of-File recClientClass.h
```

```
//*************************************************************************
//   File: recClientClass.C
//   Purpose: Gives the class definition of the Nuance RecClient class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 25 Aug 1996
//*************************************************************************
#include <iostream.h>
#include <stdlib.h>
#include <signal.h>
#include "recClientClass.h"


recClientClass *G_clientPtr;

//*************************************************************************
//     Function: recClientClass
//     Purpose:  Constructor
//     Parameters: configClass *config - Used to initialize the RecClient
//                 int timeOut - number of seconds used throughout recclient
//                               as a time out time.
//                 XtAppContext appContext - Needed for X-window app loop.
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
recClientClass::recClientClass(configClass *config, int timeOut, XtAppContext
appContext)
{
    //configPtr will be used to get the Nuance config pointer to init.
    //the recognizer
    configPtr = config -> getConfig();

    //thisObjectPtr holds a pointer to the current instance of this class
    //that will be used by a static member function used in XtAppAddInput().
    //It would not be able to get the this pointer any other way.
    recClientClassObjectPtr = this;
    G_clientPtr = this;

    //Initialize the rest of the object, and start the event processing cycle.
    initRecClient(timeOut, appContext);

    //Set the wiatingForEvent flag to false
    waitingForEvent = 0;

    cout << "recClientClass object has been instantiated." << endl;

    return;
} //End constructor
```

```
//*************************************************************************
//     Function: recClientClass
//     Purpose:  Copy constructor
//     Parameters: const recClientClass &
//     Return:   none
//     Last Date Modified: 21 Aug 1996
//*************************************************************************
recClientClass::recClientClass(const recClientClass & recClientObj)
{
    configPtr = CoronaConfigCopy(recClientObj.configPtr, &status);
    clientPtr = RCInitialize(configPtr, &status);
    status = RecResultCopy(recClientObj.resultsPtr,  resultsPtr);
    check(status, __LINE__);
    recFileDescriptor = recClientObj.recFileDescriptor;
    waitingForEvent = recClientObj.waitingForEvent;
    recClientClassObjectPtr = this;
} //End Copy constructor


//*************************************************************************
//     Function: operator =
//     Purpose:  Copy constructor
//     Parameters: const recClientClass &
//     Return:   none
//     Last Date Modified: 21 Aug 1996
//*************************************************************************
recClientClass& recClientClass::operator=(const recClientClass& recClientObj)
{
    configPtr = CoronaConfigCopy(recClientObj.configPtr, &status);
    clientPtr = RCInitialize(configPtr, &status);
    recFileDescriptor = recClientObj.recFileDescriptor;
    waitingForEvent = recClientObj.waitingForEvent;
    recClientClassObjectPtr = this;

    return (*this);
} //End operator =


//*************************************************************************
//     Function: ~recClientClass
//     Purpose:  Destructor
//     Parameters: void
//     Return:   none
//     Last Date Modified: 21 Aug 1996
//*************************************************************************
recClientClass::~recClientClass()
{
    CoronaConfigFree(configPtr);
    status = RecResultDelete(resultsPtr);
    check(status, __LINE__);

    status = RCTerminate(clientPtr);
    check(status, __LINE__);

} //End destructor
```

```
//*************************************************************************
//     Function: initRecClient
//     Purpose:  Initializes the Nuance RecClient object with error checking
//               It also starts the Nuance events processing loop.
//     Parameters: int timeOut - How many seconds will we allow the system
//                               to initialize.
//                 XtAppContext - Needed by XtAppAddInput function.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
void recClientClass::initRecClient(int timeOut, XtAppContext appContext)
{
    CoronaStatus childStatus;

    //Setup unexpected termination callback
    setupSuddenDeath();

    //Start initialization
    clientPtr = RCInitialize(configPtr, &status);
    if (status != CORONA_OK) {
       cerr << "RCInitialize routine failed with the following Nuance Msg:"
            << endl << endl
            << CoronaErrorMessage(status) << endl;
       exit(-1);
    }
    else {
       cout << "RecClient Initialization has started" << endl << endl;
    }

    //Start the process events loop
    status = RCGetFD(clientPtr, &recFileDescriptor);
    check(status, __LINE__);
    if (appContext != NULL) {
       XtAppAddInput(appContext, recFileDescriptor, (XtPointer) XtInputReadMask,
                     (XtInputCallbackProc) recClientClass::processEventsCB,
                      &recClientClassObjectPtr);
    }

    cout << "Event Processing Loop has started" << endl;

    //Setup RecResults structure.
    resultsPtr = RecResultNew();

    return;
} //End initRecClient
```

```
//*************************************************************************
//     Function: listen
//     Purpose:  Performs automatic starting/stopping of listening action.
//     Parameters: char * grammar - what grammar should be used to process
//                    input. Default = NULL -> uses grammar last used.
//                 float timeOut - how many seconds do we wait to start.
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::listen(char * grammar, float timeOut)
{
    //Start listening for beginning of speech. Use the specified grammar.
    //Speech must start by the timeout time.
    cout << "Starting to listen to speech." << endl;
    status = RCRecognize(clientPtr, grammar, timeOut);
    check(status, __LINE__);

    return;
} //End listen


//*************************************************************************
//     Function: startListening
//     Purpose:  Explicitly starts the listening process.
//     Parameters: char * grammar - what grammar should be used to process
//                    input. Default = NULL -> uses grammar last used.
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::startListening(char * grammar)
{
} //End startListening


//*************************************************************************
//     Function: stopListening
//     Purpose:  Explicitly stops the listening process.
//     Parameters: void
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::stopListening()
{
} //End stopListening
```

```
//*****************************************************************************
//     Function: abort
//     Purpose:  Stops the recognition process. Reset all result structures
//     Parameters: void
//     Return:   void
//     Last Date Modified: 10 July 1996
//*****************************************************************************
void recClientClass::abort()
{
    status = RCAbort(clientPtr);
    check(status, __LINE__);

    return;
} //End abort


//*****************************************************************************
//     Function: regCallback
//     Purpose:  Register a callback function, for a given Nuance event,
//               with the Nuance system.
//     Parameters: CoronaEvent event - One of seven events that may occur
//                 RCCallbackFnPtr CBFuncPtr - This is a pointer to the callback
//                   func. the func must have the following prototype:
//                   void CBFunction(void *user-data, CoronaEvent,
//                                   void *event-data)
//     Return:   void
//     Last Date Modified: 18 July 1996
//*****************************************************************************
void recClientClass::regCallback(CoronaEvent event, RCCallbackFnPtr CBFuncPtr)
{
    status = RCRegisterCallback(G_clientPtr->clientPtr,
                                event,
                                CBFuncPtr,
                                G_clientPtr->clientPtr);
    check(status, __LINE__);

    return;
} //End regCallback
```

```
//*************************************************************************
//     Function: playFile
//     Purpose:  Play a specific file holding a prior recorded utterance.
//     Parameters: char *file - the file name of the utterance.
//                 int timeOut - Number of seconds we will wait, both for
//                     starting the playback, and how long we will wait for
//                     the playback to finish once it has started.
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::playFile(char *file, int timeOut)
{
    void * noData;

    //Do not play file if the system is already playing a file
    if (waitingForEvent) {
       cout << "Waiting for current file to finish playing" << endl;

       //Wait for the Corona to signal that the playback is done.
       status = RCWaitForEvent(clientPtr, CORONA_EVENT_PLAYBACK_DONE,
                               (float)timeOut, noData,
                               sizeof(noData));

       //Either we time out, waiting to play the file, or we play
       //the files, and wait for the playback_done event to say
       //that the file has finished to avoid a race condition.
       if (status != CORONA_OK) {
          cerr << "RCPlayFile routine timed out with the following "
               << "Nuance Msg:" << endl << endl
               << CoronaErrorMessage(status) << endl;
          waitingForEvent = 0;
       }
    }

    //We have successfully waited, if necessary, to play. Set the
    //waitingForEvent flag, and play the file/utterance.
    waitingForEvent = 1;
    cout << "Starting the playback of last utterance. . ."
         << endl << endl;
    status = RCPlayFile(clientPtr, file);
//    check(status, __LINE__);

    //Wait for file to finish playing back.
    status = RCWaitForEvent(clientPtr, CORONA_EVENT_PLAYBACK_DONE,
                            (float)timeOut, noData,
                            sizeof(noData));
    if (status != CORONA_OK) {
       cerr << "RCPlayFile routine did not finish in the allotted time of "
            << timeOut << "seconds. Received the following Nuance msg"
            << endl << endl
            << CoronaErrorMessage(status) << endl;
    }
    else {
       cout << "Playback of file(s): " << file << " has COMPLETED."
            << endl << endl;
```

```
    }
    waitingForEvent = 0;

    return;
} //End playFile


//*************************************************************************
//     Function: playLastUtterance
//     Purpose:  Playback the last thing said to the Nuance system
//     Parameters: int timeOut - Number of seconds we will wait, both for
//                      starting the playback, and how long we will wait for
//                      the playback to finish once it has started.
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::playLastUtterance(int timeOut)
{
    void * noData;

    //Do not play file if the system is already playing a file
    if (waitingForEvent) {
       cout << "Waiting for current file to finish playing" << endl;

       //Wait for the Corona to signal that the playback is done.
       status = RCWaitForEvent(clientPtr, CORONA_EVENT_PLAYBACK_DONE,
                               (float)timeOut, noData,
                               sizeof(noData));

       //Either we time out, waiting to play the utterance, or we play
       //the utterance, and wait for the playback_done event to say
       //that it has finished to avoid a race condition.
       if (status != CORONA_OK) {
          cerr << "RCPlayFile routine timed out with the following "
               << "Nuance Msg:" << endl << endl
               << CoronaErrorMessage(status) << endl;
          waitingForEvent = 0;
          return;
       }
    }

    //We have successfully waited, if necessary, to play. Reset the
    //waitingForEvent flag, and play the file/utterance.
    waitingForEvent = 1;
    cout << "Starting the playback of last utterance. . ."
         << endl << endl;
    status = RCPlayLastUtterance(clientPtr);
//    check(status, __LINE__);

    //Wait for file to finish playing back.
    status = RCWaitForEvent(clientPtr, CORONA_EVENT_PLAYBACK_DONE,
                            (float)timeOut, noData,
                            sizeof(noData));
    if (status != CORONA_OK) {
       cerr << "RCPlayFile routine did not finish in the allotted time of "
```

```
            << timeOut << "seconds. " << endl
            << "Received the following Nuance msg:"
            << endl << endl
            << CoronaErrorMessage(status) << endl;
    }
    else {
       cout << "Playback of last utterance has COMPLETED."
            << endl << endl;
    }
    waitingForEvent = 0;

    return;
} //End playLastUtterance


//*************************************************************************
//     Function: killPlayBack
//     Purpose:  Stop the playback of any file/utterance.
//     Parameters: void
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::killPlayback()
{
    cout << "Killing the current playback, if any." << endl << endl;
    status = RCKillPlayback(clientPtr);
    check(status, __LINE__);

    return;
} //End killPlayBack


//*************************************************************************
//     Function:
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::getResults(int, void *) const
{
}


//*************************************************************************
//     Function: getRecClient
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 10 July 1996
//*************************************************************************
RecClient * recClientClass::getRecClient() const
{
    return clientPtr;
} //End getRecClient
```

```
//*************************************************************************
//     Function: setParameter
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::setParameter(int paramType, int paramValue)
{
} //End setParameter


//*************************************************************************
//     Function: setParameter
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::setParameter(int paramType, char paramValue)
{
} //End setParameter


//*************************************************************************
//     Function: getParameter
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::getParameter(int paramType, int *paramValue) const
{
} //End getParameter


//*************************************************************************
//     Function: getParameter
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 10 July 1996
//*************************************************************************
void recClientClass::getParameter(int paramType, char *paramValue) const
{
} //End getParameter
```

```
//***************************************************************************
//     Function: isWaitingForEvent
//     Purpose:  Checks if there is a Nuance background process running.
//               If so, we are waiting for an event to show that it is
//               complete. Important for sound playback routines.
//     Parameters: void
//     Return:   int - TRUE means we are waiting for an event to finish.
//     Last Date Modified: 10 July 1996
//***************************************************************************
int recClientClass::isWaitingForEvent() const
{
    return waitingForEvent;
} //End isWaitingForEvent


//***************************************************************************
//     Function:   suddenDeathCB
//     Purpose:    Kills the recClient in case of a system kill signal
//     Parameters: void
//     Return:     void
//     Last Date Modified: 23 Aug 1996
//***************************************************************************
void recClientClass::suddenDeathCB()
{
    extern recClientClass *G_clientPtr;
    cerr << "Have received an unexpected exit signal." << endl
         << "Exiting the program and deleting the recClient." << endl;
    delete G_clientPtr;
    exit(-1);
}


//***************************************************************************
//     Function: processEventsCB
//     Purpose:  This is the static member function that is passed to
//               XtAppAddInput to process non X events from the Nuance
//               system. Nuance uses a block/unblock file system to
//               signify when the input has occurred.
//     Parameters: recClientClass **recClientPtr  - pointer to a recClientClass
//               pointer. Needs to be that way for XtAppAddInput() that
//               is passing this value to this function.
//     Return:   void
//     Last Date Modified: 23 Aug 1996
//***************************************************************************
void recClientClass::processEventsCB(recClientClass **recClientPtr)
{
    (*recClientPtr) -> processEvents((*recClientPtr) -> getRecClient());

    return;
} //End processEventsCB
```

```
//*****************************************************************************
//     Function: processEvents
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 10 July 1996
//*****************************************************************************
void recClientClass::processEvents(RecClient * recPtr)
{
    if (recPtr) {
              status = RCProcessEvents(recPtr);
              if (CORONA_OK != status)
                 localFatalError("Xapp_process_rec_events",
                                 "RCProcessEvents returned '%s'\n",
                                 CoronaErrorMessage(status), NULL, NULL);
    }

    return;
} //End processEvents


//*****************************************************************************
//     Function: setupSuddenDeath
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 23 Aug 1996
//*****************************************************************************
void recClientClass::setupSuddenDeath()
{
    // Install the ^C handler.
    signal(SIGINT, suddenDeathCB);

    // Install the Bus Error handler.
    signal(SIGBUS, suddenDeathCB);

    // Install the Segmentation Fault handler.
    signal(SIGSEGV, suddenDeathCB);

    // Install the "% kill ..." (not -9) handler.
    signal(SIGTERM, suddenDeathCB);

    cerr << "The signal handlers have been registered." << endl;
    return;
}



//End-Of-File recClientClass.C
```

```
//***************************************************************************
//   File: NLClass.h
//   Purpose: Gives the class definition of the Nuance NL class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 7 July 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __NLCLASS_H
#define __NLCLASS_H

#include <nl.h>
#include <recclient.h>
#include <recresult.h>
#include "errorReportingClass.h"
#include "configClass.h"


class NLClass : protected errorReportingClass {

public:
    //Constructors and destructor
             NLClass(configClass *);   //uses config object
             NLClass(char *);          //uses package directory name
             ~NLClass();

    //Gets either all the slot names, or individual slot
    //names starting at 0.
    char **  getSlotNameList();
    char *   getSlotName(int);
    int      getLongestSlotNameLen();

    //Gets either all the values for the slots, or individual
    //values starting at slot 0, or by giving the slot name.
    char **  getSlotValueList();
    char *   getSlotValue(int);
    char *   getSlotValue(char *);
    int      getNumberOfSlots(){return numberOfSlots;};

    //Makes the NL engine interpret the results of the recognition
    //or interprets plain text.
    void     interpret(RecResult *);
    void     interpret(char *);

protected:
    int      countNumberOfSlots();
    char **  buildSlotNameList();
    CoronaConfig *configPtr;
    NLEngine *nlEngine;
    NLResult *nlResult;
    char **  slotNameList;
    int      numberOfSlots;
```

```
    int      longestSlotNameLen;
    CoronaStatus status;
};

#endif

//End-Of-File NLClass.h
```

```
//***************************************************************************
//   File: NLClass.C
//   Purpose: Gives the class definition of the Nuance NL class. This
//            encapsulates the Nuance API so that others can more easily
//            use the API in their own applications
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 20 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include "NLClass.h"


//***************************************************************************
//     Function: NLClass
//     Purpose:  Constructor
//     Parameters: configClass * config - uses configClass object with
//                 initialized information to setup NL engine and results
//     Return:   none
//     Last Date Modified: 20 Aug 1996
//***************************************************************************
NLClass::NLClass(configClass * config)
{
    //Initailize local and object variables
    configPtr = config -> getConfig();
    int nl_defined = config -> isNLDefined();
    numberOfSlots = 0;
    longestSlotNameLen = 0;
    slotNameList = NULL;

    /* Check if NL is defined and prepare accordingly */
    if (nl_defined) {
       nlEngine = NLInitializeEngine(configPtr, &status);
       check(status, __LINE__);
       nlResult = NLInitializeResult(&status);
       check(status, __LINE__);
       cout << "NLClass Object is being instantiated" << endl;

       /* NL is only defined when 1 or more slots have been defined */
       numberOfSlots = countNumberOfSlots();
       if (numberOfSlots > 0) {
          slotNameList = buildSlotNameList();
       }
       else {
          nl_defined = 0;
          cout << "However, no slots have been defined, so the" << endl
               << "NLClass object is useless!" << endl << endl;
       }
    }
    else {
        cout << "Natural Language has not been defined for this package"
```

```
             << endl << "The NLClass object just instantiated is useless!"
             << endl << endl;
    }

    CoronaConfigFree(configPtr);
} //End NLClass


//*************************************************************************
//     Function: NLClass
//     Purpose:  Constructor
//     Parameters: char * packageDir - uses package directory name to get
//               NL information needed to initialized NL engine and results.
//     Return:   none
//     Last Date Modified: 10 July 1996
//*************************************************************************
NLClass::NLClass(char *packageDir)
{
    //Initailize local and object variables
    numberOfSlots = 0;
    longestSlotNameLen = 0;
    slotNameList = NULL;

    /* Assuming that NL is defined and prepare accordingly */
    nlEngine = NLInitializeEngineFromPackageDir(packageDir, &status);
    check(status, __LINE__);
    nlResult = NLInitializeResult(&status);
    check(status, __LINE__);
    cout << "NLClass Object is being instantiated" << endl;

    /* NL is only defined when 1 or more slots have been defined */
    numberOfSlots = countNumberOfSlots();
    if (numberOfSlots > 0) {
       slotNameList = buildSlotNameList();
    }
    else {
       cout << "However, no slots have been defined, so the" << endl
            << "NLClass object is useless!" << endl << endl;
    }
} //End NLClass
```

```
//*************************************************************************
//     Function: ~NLClass
//     Purpose:  Destructor
//     Parameters: void
//     Return:   none
//     Last Date Modified: 8 July 1996
//*************************************************************************
NLClass::~NLClass()
{
    CoronaConfigFree(configPtr);
    NLFreeEngine(nlEngine);
    NLFreeResult(nlResult);

    for (int ix = 0; ix < numberOfSlots; ix++) {
              delete [] slotNameList[ix];
    }

    delete [] slotNameList;

    cerr << "NLClass Object has been destroyed." << endl;
} //End Destructor


//*************************************************************************
//     Function: getSlotNameList
//     Purpose:  gives a copy of the whole slot name list.
//        NOTE:  The char ** pointer needs to be deleted by the calling
//               program
//     Parameters: void
//     Return:   char ** list - dynamically allocated list of strings
//     Last Date Modified: 10 July 1996
//*************************************************************************
char ** NLClass::getSlotNameList()
{
    char **list = new char*[numberOfSlots];

    //Do memberwise copy of slot names
    for (int ix = 0; ix < numberOfSlots; ix++) {
       list[ix] = strdup(slotNameList[ix]);
    }

    return list;
} //End getSlotNameList
```

```
//*************************************************************************
//     Function: getSlotName
//     Purpose:  Copies an individual slot name.
//        NOTE:  The char * pointer needs to be deleted by the calling
//               program.
//     Parameters: int nthSlot - index into slot list(1st slot = 0)
//     Return:   char * - the slot name, dynamically allocated.
//     Last Date Modified: 20 Aug 1996
//*************************************************************************
char * NLClass::getSlotName(int nthSlot)
{
    char *name = strdup(slotNameList[nthSlot]);
    return name;
} //End getSlotName


//*************************************************************************
//     Function: getSlotValueList
//     Purpose:  Gets the value of ALL slots and stores them as strings.
//               Unfilled slots are NULL strings.
//        NOTE:  The char ** pointer to the strings needs to deleted,
//               along with all the strings pointed to, by the calling
//               program.
//     Parameters: void
//     Return:   char ** list - dynamically allocated list of strings
//     Last Date Modified: 10 July 1996
//*************************************************************************
char ** NLClass::getSlotValueList()
{
    const int BUFFERLEN = 100;
    char buffer[BUFFERLEN];
    char **list = new char*[numberOfSlots];

    //get value of each slot and convert into a string
    for (int ix = 0; ix < numberOfSlots; ix++) {
       status = NLGetSlotValueAsString(nlResult, slotNameList[ix], buffer,
                                       BUFFERLEN);

       //check if there is a value to copy, or if something went wrong
       if (status == CORONA_OK) {
          list[ix] = strdup(buffer);
       }
       else {
          if (status == CORONA_SLOT_NOT_FILLED) {
             list[ix] = NULL;
          }
          else {
             check(status, __LINE__);
          }
       }
    }

    return list;
} //End getSlotValueList
```

```
//*****************************************************************************
//     Function: getSlotValue
//     Purpose:  Gets the slot's value indexed by slot position(1st = 0).
//     Parameters: int nthSlot - gives the slot position whose value we want.
//     Return:   char * - value returned
//     Last Date Modified: 1 Sep 1996
//*****************************************************************************
char * NLClass::getSlotValue(int nthSlot)
{
    //Local variables
    char *value;
    NLValueType valueType;
    int tempInt;
    const int BUFFERLEN = 100;
    char buffer[BUFFERLEN];
    char slotName[BUFFERLEN];

    //Allocate enough space for the "value"
    value = new char[BUFFERLEN];
    value[0] = '\0';

    //Check if there is an nth slot
    if (nthSlot >= numberOfSlots || nthSlot < 0) {
       value = NULL;
       cerr << "Requested value for slot " << nthSlot <<":" << endl
            << "This slot index is not valid." << endl << endl;
       return value;
    }

    //Get the nth slot name and type, checking for errors along the way
    status = NLGetIthSlotNameInApplication(nlEngine, nthSlot, slotName,
                                           BUFFERLEN);
    check(status, __LINE__);

    status = NLGetSlotType(nlResult, slotName, &valueType);
    if (status == CORONA_SLOT_NOT_FILLED) {
       cerr << "Slot \"" << slotName << "\" was not filled." << endl
            << "The value returned is NULL" << endl << endl;
       value[0] = NULL;
       return value;
    }
    else {
       if (status != CORONA_OK) {
          check(status, __LINE__);
       }
    }

    switch (valueType) {
       case (NL_INT_VALUE):
          status = NLGetIntSlotValue(nlResult, slotName, &tempInt);
          check(status, __LINE__);
          sprintf(value, "%d", tempInt);
          break;
       case (NL_STRING_VALUE):
          status = NLGetStringSlotValue(nlResult, slotName, buffer, BUFFERLEN);
```

```
            check(status, __LINE__);
            strncpy(value, buffer, BUFFERLEN);
            break;
         case (NL_STRUCTURE_VALUE):
            cerr << "NLClass does not handle structures yet" << endl
                 << "Setting value of slot " << slotName << " to zero"
                 << endl << endl;
            break;
         default:
            cerr << "Encountered an unknown value type in line " << __LINE__
                 << endl << "while accessing slot " << slotName << endl
                 << "Exiting the program." << endl << endl;
            exit(-1);
     }

     return value;
} //End getSlotValue


//***************************************************************************
//     Function: getSlotValue
//     Purpose:  Gets the slot's value indexed by its name.
//     Parameters: char * nthSlot - gives the slot name whose value we want
//     Return:   char * - value returned
//     Last Date Modified: 20 Aug 1996
//***************************************************************************
char * NLClass::getSlotValue(char * nthSlot)
{
    //Local variables
    char *value;
    NLValueType valueType;
    int tempInt;
    const int BUFFERLEN = 100;
    char buffer[BUFFERLEN];
    char slotName[BUFFERLEN];

    //Allocate enough space for the "value"
    value = new char[BUFFERLEN];
    value[0] = '\0';

    //Given the slot name, find the value. We check here for errors
    status = NLGetSlotType(nlResult, slotName, &valueType);

    if (status == CORONA_SLOT_NOT_FILLED) {
       cerr << "Slot \"" << slotName << "\" was not filled." << endl
            << "The value returned is NULL" << endl << endl;
       value[0] = NULL;
       return value;
    }
    else {
       if (status != CORONA_OK) {
          check(status, __LINE__);
       }
    }
```

```
    //Given a value type, put it into the string variable 'value'
    switch (valueType) {
       case (NL_INT_VALUE):
          status = NLGetIntSlotValue(nlResult, slotName, &tempInt);
          check(status, __LINE__);
          sprintf(value, "%d", tempInt);
          break;
       case (NL_STRING_VALUE):
          status = NLGetStringSlotValue(nlResult, slotName, buffer, BUFFERLEN);
          check(status, __LINE__);
          strncpy(value, buffer, BUFFERLEN);
          break;
       case (NL_STRUCTURE_VALUE):
          sprintf(value, "%d", 0);
          cerr << "NLClass does not handle structures yet" << endl
               << "Setting value of slot " << slotName << " to zero"
               << endl << endl;
          break;
       default:
          cerr << "Encountered an unknown value type in line " << __LINE__
               << endl << "while accessing slot " << slotName << endl
               << "Exiting the program." << endl << endl;
          exit(-1);
    }

    return value;
} //End getSlotValue


//***************************************************************************
//     Function: interpret
//     Purpose:  Cause the NL engine to fill in slot values
//     Parameters: RecResult * result - the text from the recognition
//                 engine that will be interpretted to get slot values.
//     Return:   void - result is in NLResult object
//     Last Date Modified: 8 July 1996
//***************************************************************************
void NLClass::interpret(RecResult * result)
{
     status = NLInterpretRecResult(nlEngine, result, nlResult);
     check(status, __LINE__);
     return;
} //End interpret
```

```
//***************************************************************************
//     Function: interpret
//     Purpose:  Cause the NL engine to fill in slot values
//     Parameters: char * text - the text that will be interpreted to get
//                 slot values.
//     Return:   void - result is in NLResult object
//     Last Date Modified: 8 July 1996
//***************************************************************************
void NLClass::interpret(char * text)
{
     status = NLInterpretText(nlEngine, text, nlResult);
     check(status, __LINE__);
     return;
} //End interpret


//***************************************************************************
//     Function: buildSlotNameList
//     Purpose:  Utility func. that creates a list of all slot names
//               in the package.
//     Parameters: void
//     Return:   char ** list - This var. will be dynamically allocated
//     Last Date Modified: 10 July 1996
//***************************************************************************
char ** NLClass::buildSlotNameList()
{
    //Local Variables
    int  i, len;
    int  longest_len = 0;
    char slot_name_buf[1000];

    char **list = new char*[numberOfSlots];

    for (i = 0; i < numberOfSlots; i++) {
        status = NLGetIthSlotNameInApplication(nlEngine, i, slot_name_buf,
                                               1000);
        check(status, __LINE__);

        list[i] = strdup(slot_name_buf);

        /* find the longest slot name - used for formatting later on */
        len = strlen(slot_name_buf);
        if (len > longest_len){
           longest_len = len;
        }
    }

    longestSlotNameLen = longest_len;
    return list;
} //End buildSlotNameList
```

```
//*****************************************************************************
//     Function: countNumberOfSlots
//     Purpose:  Utility func. that counts the number of slots used.
//     Parameters: void
//     Return:   int - number of slots defined in the current package
//     Last Date Modified: 8 July 1996
//*****************************************************************************
int NLClass::countNumberOfSlots()
{
    char buf[100];
    int count = 0;

    do {
       status = NLGetIthSlotNameInApplication(nlEngine, count, buf, 100);

       if (status == CORONA_OK){
          count++;
       }
    } while (status != CORONA_ARGUMENT_OUT_OF_RANGE);

    return (count);
} //End countNumberOfSlots


//*****************************************************************************
//     Function: getLogestSlotNameLen
//     Purpose:  Gets the longest length of a slot name in the current
//               application. Useful for display purposes
//     Parameters: void
//     Return:   int - length of logest slot name string
//     Last Date Modified: 8 July 1996
//*****************************************************************************
int NLClass::getLongestSlotNameLen()
{
    return longestSlotNameLen;
} //End getLongestSlotNameLen


//End-Of-File NLClass.C
```

```
//***************************************************************************
//   File: recognizerClass.h
//   Purpose: Gives the class definition of the recognizerClass. This class
//            contains the configClass, recClientClass and NLClass objects
//            needed to run a voice application
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 25 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __RECOGNIZERCLASS_H
#define __RECOGNIZERCLASS_H

#include <Xm/Xm.h>
#include "configClass.h"
#include "recClientClass.h"
#include "NLClass.h"

class recognizerClass {

public:
    //Constructors - match items needed by configClass, recClientClass
    //and NLClass constructors
    recognizerClass(int *, char **, int = 1, int = 60, XtAppContext = NULL);
    recognizerClass(char *, int = 60, XtAppContext = NULL);
    recognizerClass(FILE *, char *, int = 1, int = 1, CoronaConfig * = NULL,
                    int = 60, XtAppContext = NULL);
    recognizerClass(int *, char **, char *, int, int, int,
                    CoronaConfig *, int = 60, XtAppContext = NULL);

    //Destructor
    ~recognizerClass(){};

    //Action methods for dealing with internal/external objects
    void changeConfig(FILE *, char *, int = 1, int = 1);
    void changeConfig(int *, char **, char *, int, int, int);

    //Objects needed to do the work
    configClass    config;
    recClientClass client;
    NLClass        NLProcessor;
};

#endif

//End-Of-File recognizerClass.h
```

```
//*************************************************************************
//   File: recognizerClass.C
//   Purpose: Gives the class definition of the recognizerClass. This class
//            contains the configClass, recClientClass and NLClass objects
//            needed to run a voice application
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 25 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//*************************************************************************
#include <iostream.h>
#include "recognizerClass.h"


extern recognizerClass * G_recognizerPtr;

//*************************************************************************
//     Function: recognizerClass
//     Purpose:  Constructor: configClass uses CoronaConfigFromCommandLine
//     Parameters: Used to construct the three component objects of
//                 recognizerClass.
//               int *argcPtr - Number of commandline arguments
//               char **argvPtr - Array of commandline argument strings
//               int packageRequired - TRUE = grammar package must be entered
//               int timeOut - Number of seconds to wait for init to complete.
//               XtAppContext appContext - Needed for receiving events from X.
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
recognizerClass::recognizerClass(int *argcPtr,
                                 char **argvPtr,
                                 int packageRequired,
                                 int timeOut,
                                 XtAppContext appContext)
    : config(argcPtr, argvPtr, packageRequired),
      client(&config, timeOut, appContext),
      NLProcessor(&config)
{
    G_recognizerPtr = this;
    cout << "recognizerClass object is instantiated." << endl << endl;
} //End recognizerClass
```

```
//***************************************************************************
//     Function: recognizerClass
//     Purpose:  Constructor: configClass uses CoronaConfigFromPackageDir
//     Parameters: Used to construct the three component objects of
//                 recognizerClass.
//               char *packageDir - where can grammar package be found
//               int timeOut - Number of seconds to wait for init to complete.
//               XtAppContext appContext - Needed for receiving events from X.
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
recognizerClass::recognizerClass(char *packageDir,
                                 int timeOut,
                                 XtAppContext appContext)
    : config(packageDir),
      client(&config, timeOut, appContext),
      NLProcessor(&config)
{
    G_recognizerPtr = this;
    cout << "recognizerClass object is instantiated." << endl << endl;
} //End recognizerClass


//***************************************************************************
//     Function: recognizerClass
//     Purpose:  Constructor: configClass uses CoronaConfigFromFile
//     Parameters: Used to construct the three component objects of
//                 recognizerClass.
//               FILE * fileFD
//               char *paramSource
//               int fromUser
//               int mustBeValid
//               CoronaConfig * ConfigPtr
//               int timeOut - Number of seconds to wait for init to complete.
//               XtAppContext appContext - Needed for receiving events from X.
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
recognizerClass::recognizerClass(FILE * fileFD,
                                 char *paramSource,
                                 int fromUser,
                                 int mustBeValid,
                                 CoronaConfig * ConfigPtr,
                                 int timeOut,
                                 XtAppContext appContext)
    : config(fileFD, paramSource, fromUser, mustBeValid, ConfigPtr),
      client(&config, timeOut, appContext),
      NLProcessor(&config)
{
    G_recognizerPtr = this;
    cout << "recognizerClass object is instantiated." << endl << endl;
} //End recognizerClass
```

```
//***************************************************************************
//     Function: recognizerClass
//     Purpose:  Constructor: configClass uses CoronaConfigFromStringArray
//     Parameters: Used to construct the three component objects of
//                  recognizerClass.
//               int *stringCount
//               char **string
//               char *param_source
//               int fromUser
//               int mustBeValid
//               int removeArgs
//               CoronaConfig *ConfigPtr
//               int timeOut - Number of seconds to wait for init to complete.
//               XtAppContext appContext - Needed for receiving events from X.
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
recognizerClass::recognizerClass(int *stringCount,
                                 char **string,
                                 char *param_source,
                                 int fromUser,
                                 int mustBeValid,
                                 int removeArgs,
                                 CoronaConfig *ConfigPtr,
                                 int timeOut,
                                 XtAppContext appContext)
    : config(stringCount, string, param_source, fromUser, mustBeValid,
             removeArgs, ConfigPtr),
      client(&config, timeOut, appContext),
      NLProcessor(&config)
{
    G_recognizerPtr = this;
    cout << "recognizerClass object is instantiated." << endl << endl;
} //End recognizerClass


//***************************************************************************
//     Function: changeConfig
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 21 Aug 1996
//***************************************************************************
void recognizerClass::changeConfig(FILE *, char *, int, int)
{
} //End changeConfig
```

```
//***************************************************************************
//     Function: changeConfig
//     Purpose:
//     Parameters:
//     Return:   void
//     Last Date Modified: 21 Aug 1996
//***************************************************************************
void recognizerClass::changeConfig(int *, char **, char *, int, int, int)
{
} //End changeConfig



//End-Of-File recognizerClass.C
```

```
//***************************************************************************
//   File: voiceVkApp.h
//   Purpose: Derives a new class from the VkApp class from ViewKit. It
//            will hold the classes for the Nuance applications.
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 26 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __VOICEVKAPP_H
#define __VOICEVKAPP_H
#include <Xm/Xm.h>
#include <Vk/VkApp.h>
#include <Vk/VkResource.h>
#include "recognizerClass.h"
#include "voiceNetManagerClass.h"

class voiceVkApp : public VkApp {
public:
          voiceVkApp(char *appClassName,
                     int  *arg_c,
                     char **arg_v,
                     XrmOptionDescRec  *optionList = NULL,
                     int  sizeOfOptionList = 0);
         ~voiceVkApp();
    void  terminate(int = 0);
    virtual void  displayNLSlotsAndValues();
    virtual void  createNewRecognizer(int *, char **, int = 1, int = 60,
                                      XtAppContext = NULL);
    int   isPackageLoaded();
    int   isNetworkOpen();
    void  setNetworkStatus(int);
    int   loadPackage(char *);
    virtual int   openNetwork();
    recognizerClass      *recognizer;
    voiceNetManagerClass *netManager;

protected:
    static void processDiedCB(void *, CoronaEvent, void *);
    static void initCompleteCB(void *, CoronaEvent, void *);
    static void startOfSpeechCB(void *, CoronaEvent, void *);
    static void endOfSpeechCB(void *, CoronaEvent, void *);
    static void partialResultCB(void *, CoronaEvent, void *);
    static void finalResultCB(void *, CoronaEvent, void *);
    int         isResultStringEmpty(char *);
    void        appendDisplayMeaning(char *);
    void        displayNLSlots(void);
    int         networkOpen;
    int         packageLoaded;
    int        *argCount;
    char      **argVar;
};
#endif
//End-Of-File voiceVkApp.h
```

```
//***************************************************************************
//   File: voiceVkApp.C
//   Purpose: Derives a new class from the VkApp class from ViewKit. It
//            will hold the classes for the Nuance applications.
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 1 Sep 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#include <iostream.h>
#include <VkEZ.h>
#include <Xm/List.h>
#include "VkwindowMainWindow.h"
#include "VoiceAppDisplayClass.h"
#include "voiceVkApp.h"

//Global pointers to major object components. Space
//allocated in main.C
extern VoiceAppDisplayClass *G_mainPanelPtr;
extern recognizerClass      *G_recognizerPtr;
extern voiceVkApp           *G_appPtr;
extern voiceNetManagerClass *G_netManagerPtr;
extern VkwindowMainWindow   *G_mainWindowPtr;


//***************************************************************************
//     Function: voiceVkApp
//     Purpose:  Constructor
//     Parameters: Same params as VkApp Class
//     Return:   none
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
voiceVkApp::voiceVkApp(char *appClassName,
                       int   *arg_c,
                       char **arg_v,
                       XrmOptionDescRec  *optionList,
                       int    sizeOfOptionList)
           :VkApp(appClassName, arg_c, arg_v, optionList, sizeOfOptionList)
{
    G_appPtr = this;
    theApplication = this;
    argCount = arg_c;
    argVar = arg_v;

    //Check if the -package Nuance arg was passed in.
    //If so, create the recognizer object.
    int arg_upto = 0;
    while (arg_upto < *arg_c) {
       if (!strcmp(arg_v[arg_upto], "-package")) {
           recognizer = new recognizerClass(arg_c, arg_v,
                                            1, 60, appContext());
           packageLoaded = TRUE;
           break;
       }
```

```
        else {
            packageLoaded = FALSE;
         }
         arg_upto++;
    }

    //If package was not loaded, desensitize button on the voice panel.
    if (!packageLoaded) {
       XtSetSensitive(G_mainPanelPtr->_listenButton, FALSE);
       XtSetSensitive(G_mainPanelPtr->_playbackButton, FALSE);
       XtSetSensitive(G_mainPanelPtr->_abortbutton, FALSE);
    }

    //Setup the callbacks for the SLU system
    if (packageLoaded) {
       recognizer->client.regCallback(CORONA_EVENT_PROCESS_DIED, processDiedCB);
       recognizer->client.regCallback(CORONA_EVENT_INIT_COMPLETE,
                                      initCompleteCB);
       cout << "The callbacks for PROCESS_DIED and INIT_COMPLETE have been reg."
            << endl;
    }
} //End voiceVkApp


//****************************************************************************
//     Function: terminate
//     Purpose:  Overload the VkApp terminate func. that provides a clean
//               exit of the application
//     Parameters:
//     Return:
//     Last Date Modified: 25 Aug 1996
//****************************************************************************
void voiceVkApp::terminate(int status)
{
    cerr << "The voiceVkApp destructor has finished" << endl;

    VkApp::terminate(status);
    return;
} //End terminate


//****************************************************************************
//     Function: ~voiceVkApp
//     Purpose:  destructor
//     Parameters: void
//     Return:   none
//     Last Date Modified: 28 Aug 1996
//****************************************************************************
voiceVkApp::~voiceVkApp()
{
    terminate();
}
```

```
//**************************************************************************
//     Function: createNewRecognizer
//     Purpose:  Creates a new recognizer object
//     Parameters: Used to construct the three component objects of
//                 recognizerClass.
//               int *argcPtr - Number of commandline arguments
//               char **argvPtr - Array of commandline argument strings
//               int packageRequired - TRUE = grammar package must be entered
//               int timeOut - Number of seconds to wait for init to complete.
//               XtAppContext appContext - Needed for receiving events from X.
//     Return:   none
//     Last Date Modified: 30 Aug 1996
//**************************************************************************
void voiceVkApp::createNewRecognizer(int *arg_c,
                                     char **arg_v,
                                     int packageRequired,
                                     int timeOut,
                                     XtAppContext appContext)
{
    if (packageLoaded) {
       delete recognizer;
       if (networkOpen) {
          delete netManager;
       }
    }

    argCount = arg_c;
    argVar = arg_v;

    //Check if the -package Nuance arg was passed in.
    //If so, create the recognizer object.
    int arg_upto = 0;
    while (arg_upto < *arg_c) {
       if (!strcmp(arg_v[arg_upto], "-package")) {
          recognizer = new recognizerClass(arg_c, arg_v,
                                           1, 60, appContext);
          packageLoaded = TRUE;
          break;
        }
        else {
           packageLoaded = FALSE;
        }
        arg_upto++;
    }

    //Set up the waiting screen so the user cannot interrupt initialization
    //Will become "unbusy" in initCompleteCB
    busy("Initializing the Recognition\nServer...\n\nThis may take 30 seconds.",
         G_mainPanelPtr->_parent);

    //If the network had been up, bring it up again.
    if (networkOpen) {
              networkOpen = FALSE;
              networkOpen = openNetwork();
    }
```

```
    //Make the radio button reflect the current status
    Widget w = *(G_mainWindowPtr->_option1);
    EZ(w) = networkOpen;

    //If package was not loaded, desensitize button on the voice panel.
    if (!packageLoaded) {
       XtSetSensitive(G_mainPanelPtr->_listenButton, FALSE);
       XtSetSensitive(G_mainPanelPtr->_playbackButton, FALSE);
       XtSetSensitive(G_mainPanelPtr->_abortbutton, FALSE);
    }
    //If the package was load, sensitize the buttons.
    else {
       XtSetSensitive(G_mainPanelPtr->_listenButton, TRUE);
       XtSetSensitive(G_mainPanelPtr->_playbackButton, TRUE);
       XtSetSensitive(G_mainPanelPtr->_abortbutton, TRUE);
    }

    //Setup the callbacks for the SLU system
    if (packageLoaded) {
       recognizer->client.regCallback(CORONA_EVENT_PROCESS_DIED, processDiedCB);
       recognizer->client.regCallback(CORONA_EVENT_INIT_COMPLETE,
                                      initCompleteCB);
       cout << "The callbacks for PROCESS_DIED and INIT_COMPLETE have been reg."
            << endl;
    }
} //End createNewRecognizer



//***************************************************************************
//     Function: isNetworkOpen
//     Purpose:  Tells if the netManager is up and running
//     Parameters: void
//     Return:   int - TRUE if netManager is up
//     Last Date Modified: 28 Aug 1996
//***************************************************************************
int voiceVkApp::isNetworkOpen()
{
    return networkOpen;
} //End isNetworkOpen


//***************************************************************************
//     Function: setNetworkStatus
//     Purpose:  Sets the status of the voiceNetManager - up or down
//     Parameters: int - Zero = down, non-zero is up
//     Return:   int - TRUE if netManager is up
//     Last Date Modified: 1 Sep 1996
//***************************************************************************
void voiceVkApp::setNetworkStatus(int netStatus)
{
    networkOpen = netStatus;
} //End setNetworkStatus
```

```
//***************************************************************************
//     Function: isPackageLoaded
//     Purpose:  Has a packagebeen loaded and the recognizer been created
//     Parameters: void
//     Return:   int - TRUE means that the recognizer is up and running
//     Last Date Modified: 28 Aug 1996
//***************************************************************************
int voiceVkApp::isPackageLoaded()
{
    return packageLoaded;
}


//***************************************************************************
//     Function: openNetwork
//     Purpose:  Tells if the netManager is up and running
//     Parameters: void
//     Return:   int - TRUE if netManager is up
//     Last Date Modified: 1 Sep 1996
//***************************************************************************
int voiceVkApp::openNetwork()
{

    if (networkOpen == FALSE && packageLoaded == TRUE) {
       netManager = new voiceNetManagerClass(argCount,
                                             argVar,
                                             recognizer,
                                             12,  //Num of fields in IDU
                                             14); //Len of field in IDU
    }

    //Check if the network was openned.
    if (netManager) {
       networkOpen = TRUE;
       cerr << "The IDU Net is openned." << endl << endl;
    }
    else {
       networkOpen = FALSE;
       cerr << "The IDU Net was not openned. This could be because." << endl
            << "the network failed to open,  or a grammar package" << endl
            << "has not been loaded." << endl << endl;
    }

    //Set the global variable G_netManagerPtr to point to voiceNetManager
    //object
    G_netManagerPtr = netManager;

    return networkOpen;
} //End isNetworkOpen
```

```
//
//  All the following Callback Functions are static member functions.
//


//*************************************************************************
//     Function: processDiedCB
//     Purpose:  Callback when an unexpected interrupt kills the application,
//               or the recognizer(specifically the recClientClass obj.) dies.
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
void voiceVkApp::processDiedCB(void *rec, CoronaEvent event, void *eventData)
{
    extern recognizerClass *G_recognizerPtr;

    cerr << "The recClient process died." << endl;
    delete G_recognizerPtr;

    return;
} //End processDiedCB


//*************************************************************************
//     Function: initCompleteCB
//     Purpose:  Callback that registers the callbacks for all the
//               other Corona Events that may happen in the application
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
void voiceVkApp::initCompleteCB(void *rec, CoronaEvent event, void *eventData)
{

    extern recognizerClass *G_recognizerPtr;

    G_recognizerPtr->client.regCallback(CORONA_EVENT_START_OF_SPEECH,
                                        startOfSpeechCB);
    G_recognizerPtr->client.regCallback(CORONA_EVENT_END_OF_SPEECH,
                                        endOfSpeechCB);
    G_recognizerPtr->client.regCallback(CORONA_EVENT_PARTIAL_RESULT,
                                        partialResultCB);
    G_recognizerPtr->client.regCallback(CORONA_EVENT_FINAL_RESULT,
                                        finalResultCB);

    //Initialize the grammar list on the status conrol panel
    char **grammars = G_recognizerPtr->config.getGrammarNames();
    int number = G_recognizerPtr->config.getNumberOfGrammars();
```

```
    EZ(G_mainPanelPtr->_statusClass->_statusTextfield) = "Ready";
    XmString item;
    if (number != 0) {
       XmListDeleteAllItems(G_mainPanelPtr->_statusClass->_grammarListField);
       item = XmStringCreateLtoR(grammars[0], XmFONTLIST_DEFAULT_TAG);
       XmListAddItemUnselected(
            (G_mainPanelPtr->_statusClass->_grammarListField),
            item,
            1);
       XmStringFree(item);
    }

    int ix = 2;
    while (ix <= number) {
       item = XmStringCreateLtoR(grammars[ix - 1], XmFONTLIST_DEFAULT_TAG);
       XmListAddItemUnselected(
            (G_mainPanelPtr->_statusClass->_grammarListField),
            item,
            ix++);
       XmStringFree(item);
    }

    //Delete the temporary grammar list
    for (ix = 0; ix < number; ix++) {
       delete [] grammars[ix];
    }
    delete [] grammars;

    cout << "The Corona Event callbacks have been registered." << endl
         << "Exiting initCompleteCB." << endl;

    G_appPtr->notBusy();
    return;
} //End initCompleteCB


//*************************************************************************
//     Function: startOfSpeechCB
//     Purpose:  Callback for the start-of-speech Nuance event
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
void voiceVkApp::startOfSpeechCB(void *rec, CoronaEvent event, void *eventData)
{
    cout << "The start-of-speech callback has been called." << endl;

    EZ(G_mainPanelPtr->_statusClass->_statusTextfield) = "Listening...";
    EZ(G_mainPanelPtr->_recDisplayClass->_textfield) = "\0";

    return;
} //End startOfSpeechCB
```

```
//***************************************************************************
//     Function: endOfSpeechCB
//     Purpose:  Callback for the end-of-speech Nuance event
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
void voiceVkApp::endOfSpeechCB(void *rec, CoronaEvent event, void *eventData)
{
    cout << "The end-of-speech callback has been called." << endl;

    EZ(G_mainPanelPtr->_statusClass->_statusTextfield) = "Stopped Listening";
    return;
} //End endOfSpeechCB


//***************************************************************************
//     Function: partialResultCB
//     Purpose:  Callback for the partial-result Nuance event
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//***************************************************************************
void voiceVkApp::partialResultCB(void *rec, CoronaEvent event, void *eventData)
{
    char result[1000];

    EZ(G_mainPanelPtr->_statusClass->_statusTextfield) = "Calculating";

    G_recognizerPtr->client.resultsPtr = (RecResult *)eventData;
    RecResultString(G_recognizerPtr->client.resultsPtr, 0, result,
                    sizeof(result));

    if (G_appPtr->isResultStringEmpty(result)){
       result[0] = NULL;
       EZ(G_mainPanelPtr->_recDisplayClass->_textfield) = result;
    }
    else{
       EZ(G_mainPanelPtr->_recDisplayClass->_textfield) = result;
    }

    if (G_recognizerPtr->config.isNLDefined()) {
       G_recognizerPtr->NLProcessor.interpret(G_recognizerPtr->
                                              client.resultsPtr);
       G_appPtr->displayNLSlotsAndValues();
    }

    return;
} //End partialResultCB
```

```
//*****************************************************************************
//     Function: finalResultCB
//     Purpose:  Callback for the final-result Nuance event
//     Parameters: Mandatory signature of a Nuance callback function.
//               void * rec - RecClient address
//               CoronaEvent event - Reason for the callback.
//               void *eventData - structure that contains further event data.
//     Return:   void
//     Last Date Modified: 25 Aug 1996
//*****************************************************************************
void voiceVkApp::finalResultCB(void *rec, CoronaEvent event, void *eventData)
{
    static int cmdCount = 1;
    char result[1000];

    //Set the status panel to read "Ready"
    EZ(G_mainPanelPtr->_statusClass->_statusTextfield) = "Ready";

    //Put the results into a string for output
    G_recognizerPtr->client.resultsPtr = (RecResult *)eventData;
    RecResultString(G_recognizerPtr->client.resultsPtr, 0, result,
                  sizeof(result));

    //Print out the results into the recDisplay. If the string is
    //empty, then the speech was rejected.
    int emptyResults = G_appPtr->isResultStringEmpty(result);
    if (emptyResults){
       EZ(G_mainPanelPtr->_recDisplayClass->_textfield) = "<Rejected!>";
    }
    else{
       EZ(G_mainPanelPtr->_recDisplayClass->_textfield) = result;
    }

    //If there is NLP and the speech was not rejected, give the meaning results.
    //and send out the packet if the IDU network is open.
    if (G_recognizerPtr->config.isNLDefined() && !emptyResults) {
       G_recognizerPtr->NLProcessor.interpret(G_recognizerPtr->
                                              client.resultsPtr);
       G_appPtr->displayNLSlotsAndValues();

       //If the IDU network is open, send out the packet
       if (G_appPtr->isNetworkOpen()) {
          if (G_netManagerPtr->getNLData()) {
             G_netManagerPtr->sendData();
          }
          else {
             cerr << "Could not get the NLP data from recognizer."
                  << endl;
          }
       }
    }

    //Put the recResults into the command history panel.
    XmString item = XmStringCreateLtoR(result, XmFONTLIST_DEFAULT_TAG);
    XmListAddItemUnselected(
```

```
         (G_mainPanelPtr->_recDisplayClass->_cmdHistoryScrolledList),
         item,
         cmdCount++);
    XmStringFree(item);

    return;
} //End finalResultCB


//*************************************************************************
//     Function: isResultStringEmpty
//     Purpose:  Is the string empty?
//     Parameters: char *results - string to check
//     Return:   int - 1 = true = empty
//     Last Date Modified: 25 Aug 1996
//*************************************************************************
int voiceVkApp::isResultStringEmpty(char *results)
{
    char dup[1000];
    char *ptr;

    if (!results){
       return 1;
    }

    strncpy(dup, results, sizeof(dup));
    ptr = (char *)strtok(dup, " \n\t\r");
    if (!ptr) {
       return 1;
    }
    else{
       return 0;
    }
} //End isResultStringEmpty


//*************************************************************************
//     Function: appendDisplayMeaning
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 26 Aug 1996
//*************************************************************************
void voiceVkApp::appendDisplayMeaning(char *text)
{
    EZ(G_mainPanelPtr->_nLDisplayClass->_nlScrolledText) << text;

    return;
} //End appendDisplayMeaning
```

```
//*****************************************************************************
//     Function: displayNLSlots
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 26 Aug 1996
//*****************************************************************************
void voiceVkApp::displayNLSlots()
{
    int ix;
    char buf[1000];
    int num_slots = recognizer->NLProcessor.getNumberOfSlots();
    int longest_slot_name_len = recognizer->
                                NLProcessor.getLongestSlotNameLen();
    char **slot_name_list = recognizer->NLProcessor.getSlotNameList();

    // clear the meaning text field
    EZ(G_mainPanelPtr->_nLDisplayClass->_nlScrolledText) =  "\0";

    // display the slot names
    for (ix = 0; ix < num_slots; ix++) {

              // normalize length of slot name to equal longest name
              sprintf(buf, "%*s", longest_slot_name_len, slot_name_list[ix]);
              appendDisplayMeaning(buf);
              appendDisplayMeaning(":");

              // don't display a new line on last entry
              if (ix + 1 < num_slots) {
                  appendDisplayMeaning("\n");
              }
    }

    //Delete this temp dynamic memory
    for (ix = 0; ix < num_slots; ix++){
              delete [] slot_name_list[ix];
    }
    delete [] slot_name_list;

    return;
} //End displayNLSlots
```

```
//*****************************************************************************
//     Function: displayNLSlotsAndValues
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 26 Aug 1996
//*****************************************************************************
void voiceVkApp::displayNLSlotsAndValues()
{
    int ix;
    char buf[1000];
    CoronaStatus status;
    int num_slots = recognizer->NLProcessor.getNumberOfSlots();
    int longest_slot_name_len = recognizer->
                                NLProcessor.getLongestSlotNameLen();
    char **slot_name_list = recognizer->NLProcessor.getSlotNameList();
    char **slot_value_list = recognizer->NLProcessor.getSlotValueList();

    /* clear the meaning text field */
    EZ(G_mainPanelPtr->_nLDisplayClass->_nlScrolledText) = "\0";

    /* display the slot names & values */
    for (ix = 0; ix < num_slots; ix++) {
        /* normalize length of slot name to equal longest name */
        sprintf(buf, "%*s", longest_slot_name_len, slot_name_list[ix]);
        appendDisplayMeaning(buf);
        appendDisplayMeaning(":");

        /* display the slot value */
        appendDisplayMeaning(slot_value_list[ix]);

        /* don't dispaly new line on last entry */
        if (ix + 1 < num_slots) {
           appendDisplayMeaning("\n");
        }
    }

    //Delete this temp dynamic memory
    for (ix = 0; ix < num_slots; ix++){
        delete [] slot_name_list[ix];
        delete [] slot_value_list[ix];
    }
    delete [] slot_name_list;
    delete [] slot_value_list;

    return;
} //End displayNLSlotsAndValues



//End-Of-File voiceVkApp.C
```

```
//***************************************************************************
//   File: voiceNetManagerClass.h
//   Purpose: Declares the class that will handle translating the NLP
//            results into a packet that will be sent out on the IDU net.
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 27 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#ifndef __VOICENETMANAGERCLASS_H
#define __VOICENETMANAGERCLASS_H

#include <Vk/VkApp.h>
#include <Vk/VkResource.h>
#include <idunetlib.h>
#include <idu.h>
#include "recognizerClass.h"


class voiceNetManagerClass {
public:
          voiceNetManagerClass(int *,
                               char **,
                               recognizerClass *,
                               int,
                               int);
         ~voiceNetManagerClass();
    int   getNLData();
    int   sendData();

protected:
    int                  initComm(int, char **);
    void                 clearIDU();
    IDU_net_manager     *net;
    VoiceAppToNPSNETIDU  voiceidu;
    recognizerClass     *recognizer;
    int                  num_voiceidu_fields;
    int                  len_voiceidu_field;
};

#endif


//End-Of-File voiceNetManagerClass.h
```

```
//***************************************************************************
//   File: voiceNetManagerClass.C
//   Purpose: Declares the class that will handle translating the NLP
//            results into a packet that will be sent out on the IDU net.
//   Environment: SGI
//   Operating System: Irix 6.2
//   Author: Capt Edward M. DeVilliers
//   Last Date Modified: 27 Aug 1996
//   Copyright 1996, Naval Postgraduate School, NPSNET Research Group
//***************************************************************************
#include <iostream.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include "voiceNetManagerClass.h"


//***************************************************************************
//     Function: voiceNetManagerClass
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//***************************************************************************
voiceNetManagerClass::voiceNetManagerClass(int *argc, char **argv,
                                           recognizerClass *rec,
                                           int num, int len)
             : num_voiceidu_fields(num),
               len_voiceidu_field(len),
               recognizer(rec)
{
    enum Outcome {NET_OPEN_FAILED, BAD_USAGE, SUCCESSFUL};
    int appNameLength = 14;

    int status = initComm(*argc, argv);
    switch (status) {
       case NET_OPEN_FAILED:
          cerr << "IDU Net could not be openned." << endl;
          break;
       case BAD_USAGE:
          cerr << "Usage:  idudump [-p <network port>] \n"
               << "                [-i <network interface>] \n"
               << "                [-g <multicast group>] \n"
               << "                [-t <multicast ttl>] \n"
               << "                [-b   (to enable broadcast) "
               << endl;
          break;
       case SUCCESSFUL:
          cerr << "The IDU Net has been openned successfully." << endl;
          break;
       default:
          cerr << "Unknown return value passed from InitComm." << endl
               << "Exiting the program." << endl;
          exit(-1);
          break;
```

```
    }

    strncpy(voiceidu.appName, "NPSNET", appNameLength);
} //End voiceNetManagerClass


//***************************************************************************
//     Function: ~voiceNetManagerClass
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//***************************************************************************
voiceNetManagerClass::~voiceNetManagerClass()
{
    net->net_close();
    delete net;
} //End ~voiceNetManagerClass


//***************************************************************************
//     Function: getNLData
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//***************************************************************************
int voiceNetManagerClass::getNLData()
{
    char *tempData;
    int num_slots = recognizer->NLProcessor.getNumberOfSlots();
    int length = 0;

    //If the grammar has too many slots, this routine will not
    //produce the correct results.
    if (num_slots > num_voiceidu_fields) {
       cerr << "The current grammar has too many NL slots defined." << endl
            << "Currently, only " << num_voiceidu_fields << " are allowed."
            << endl;
       return FALSE;
    }

    //Copy the slot values into the voiceidu
    for (int ix = 0; ix < num_slots; ix++) {
        tempData = recognizer->NLProcessor.getSlotValue(ix);
        length = strlen(tempData);
        cerr << "Got a slot value of :" << tempData;
        //If the value is bigger than the IDU is designed to hold
        //return unsuccessfully.
        if (length > len_voiceidu_field) {
           cerr << "Slot value is longer than the IDU field can hold." << endl
                << "Getting NLP data for IDU transmission unsuccessful."
                << endl << endl;
           return FALSE;
        }
```

```
        //Copy the string value into the IDU field
        strncpy(&voiceidu.data[ix*(len_voiceidu_field + 1)], tempData,
               (len_voiceidu_field + 1));

        delete tempData;
    }


    return TRUE;
} //End getNLData


//*****************************************************************************
//     Function: sendData
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//*****************************************************************************
int voiceNetManagerClass::sendData()
{
    if ( !(net->write_idu((char *)&voiceidu, VoiceApp_To_NPSNET_Type)) ){
       cerr << "ERROR:\tCould not send voice IDU." << endl;
       return FALSE;
    }
    else {
       return TRUE;
    }
} //End sendData


//*****************************************************************************
//     Function: initComm
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//*****************************************************************************
int voiceNetManagerClass::initComm(int argc, char **argv)
{
    enum  Outcome {NET_OPEN_FAILED, BAD_USAGE, SUCCESSFUL};

    //COMMS-----------------------------------------------
    int op = 0 ;
    extern char *optarg;
    extern int optind, operr;

    // Multicast Defaults
    int     multicast = TRUE;
    u_short port = 0;
    char    group[25];
    u_char  ttl = IDU_DEF_MC_TTL;
    int     loopback = FALSE;
```

```
char net_interface[20];

//initialize voice IDU structures
clearIDU();

//COMMS-----------------------------------------------
strncpy ( group, IDU_DEF_MC_GROUP,25 );   //COMMS
strcpy ( net_interface, "" );             //COMMS

//COMMS-----------------------------------------------
while ((op = getopt(argc, argv, "P:p:G:g:T:t:BblLI:i:")) != -1)
   {
   switch (op) {
      case 'p':
      case 'P':
         port = u_short(atoi(optarg));
         break;
      case 'G':
      case 'g':
         strncpy ( group, optarg, 25 );
         break;
      case 't':
      case 'T':
         ttl =  u_char(atoi(optarg));
         break;
      case 'b':
      case 'B':
         multicast = FALSE;
         break;
      case 'l':
      case 'L':
         loopback = TRUE;
         break;
      case 'i':
      case 'I':
         strncpy ( net_interface, optarg, 19 );
         break;
      default:
         return BAD_USAGE;
      }
   }

if ( multicast ) {
   if ( port == 0 ) {
      port = IDU_DEF_MC_PORT;
   }
   net = new IDU_net_manager ( group, port, ttl, net_interface,
                               loopback );
}
else {
   if ( port == 0 ) {
      port = IDU_DEF_BC_PORT;
   }
   net = new IDU_net_manager ( port, net_interface, loopback );
}
```

```
    if ( !net->net_open() ) {
       cerr << "Could not open network." << endl;
       return NET_OPEN_FAILED;
    }

    if ( multicast ) {
       cerr << "\tMode:     \tMulticast" << endl;
       cerr << "\tPort:     \t" << (int)port << endl;
       cerr << "\tGroup:    \t" << group << endl;
       cerr << "\tTTL:      \t" << (int)ttl << endl;
       cerr << "\tInterface:\t" << net_interface << endl;
       cerr << "\tLoopback: \t";
       if ( loopback ) {
          cerr << "ON" << endl;
       }
       else {
          cerr << "OFF" << endl;
       }
       cerr << endl;
    }
    else {
       cerr << "\tMode:     \tBroadcast" << endl;
       cerr << "\tPort:     \t" << (int)port << endl;
       cerr << "\tInterface:\t" << net_interface << endl << endl;
       cerr << "\tLoopback: \t";
       if ( loopback ) {
          cerr << "ON" << endl;
       }
       else {
          cerr << "OFF" << endl;
       }
       cerr << endl;
    }
    //COMMS----------------------------------------------

    return SUCCESSFUL;
} //End initComm
```

```
//*************************************************************************
//     Function: clearIDU
//     Purpose:
//     Parameters:
//     Return:
//     Last Date Modified: 27 Aug 1996
//*************************************************************************
void voiceNetManagerClass::clearIDU()
{
    const char SPACE = '\0';
    const int SIZE = len_voiceidu_field * num_voiceidu_fields;

    for (int ix = 0; ix < SIZE; ix++) {
              voiceidu.data[ix] = SPACE;
    }

    return;
} //End clearIDU


//End-Of-File voiceNetManagerClass.C
```

LIST OF REFERENCES

[ALLE95] Allen, James, *Natural Language Understanding*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA 1995.

[ARON85] Aronoff, Mark, *Word Formation in Generative Grammar*, MIT Press, Cambridge, Massachusetts, 1985.

[BACO95] Bacon, Daniel Keith, Jr. "Integration of a Submarine into NPSNET," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[BAKE75] Baker, James, "Stochastic Modeling for Automatic Speech Understanding," *Speech Recognition*, Academic Press, London, 1975.

[BRAT96] Bratt, Harry, et al., "Technical Overview of the CommandTalk System" SRI Internal Document, January 30, 1996.

[CARB94] Carbonell, Jaime G., and Hayes, Philip J., "Natural Language Understanding," *Encyclopedia of Artificial Intelligence*, New York, 1994.

[CATE84] Cater, John P., *Electronically Hearing: Computer Speech Recognition*, Howard W. Sams & Company, Indianapolis, IN, 1984.

[COHE94] Cohen, Philip R., Oviatt, Sharon L., "The Role of Voice Input for Human-Machine Communication", Dept of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, 1994.

[CSD96] Computer Science Department, "NPSNET IV.9 User Guide," Naval Postgraduate School, 1996.

[ERMA80] Erman, Lee, et al., "The Hearsay-II Speech Understanding Sytem: Integrating Knowledge to Resolve Uncertainty," Computing Surveys, 12(2), 1980.

[EVEA96] Everett, Stephanie S., Wauchope, Kenneth, and Perez Manuel A., "A Natural Language Interface for Virtual Reality Systems," *http://www.aic.nrl.navy.mil/~severett/*, 1996.

[EVEB92] Everett, Stephanie S., Wauchope, Kenneth, and Perzanowski, Dennis, "Talking to a Natural Language Interface: Lessons Learned," *http://www.aic.nrl.navy.mil/papers/1992/AIC-92-016.ps*, Navy Center for Applied Research in Artificial Intelligence Naval Research Laboratory, 1992.

[EVEC93] Everett, Stephanie S., Wauchope, Kenneth, and Perzanowski, Dennis, “Adding Speech Recognition to a Natural Language Interface,” *http://www.aic.nrl.navy.mil/papers/1993/AIC-93-026.ps*, Navy Center for Applied Research in Artificial Intelligence Naval Research Laboratory, 1993.

[FILL68] Fillmore, Charles, “The Case for Case,” Universals in Linguistic Theory, Holt, Reinhardt, and Winston, New York, 1968.

[GRIS86] Grishman, Ralph, “PROTEUS Parser Reference Manual,” PROTEUS Project Memorandum #4, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, July 1986.

[IST93] Institute for Simulation and Training, *Standard for Information Technology - Protocols for Distributed Interactive Simulaton Applications (Draft)*, Orlando, FL, 1993.

[FALLS85] Fallside, Frank, Woods, William A., *Computer Speech Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[GUTH96] Guthrie, Louise, et al., “The Role of Lexicons in Natural Language Processing,” *Communications of the ACM*, Vol. 39, No. 1, January 1996.

[KIRB95] Kirby, Samuel A. "NPSNET: Software Requirements for Implementation of a Sand Table in the Virtual Environment," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[LCS95] Laboratory of Computer Science, *Annual Research Summary: Spoken Language Systems*, Massachusetts Institute of Technology, 1995.

[LEE89] Lee, Kai-Fu, *Automatic Speech Recognition: The Development of the SPHINX System*, Kluwer Academic Publishers, Boston, 1989.

[LENT95] Lentz, Frederick Charles, III "Integration of ASW Helicopter Operations and Environment into NPSNET," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[LOCK94] Locke, John, “An Introduction to the Internet Networking Environment and SIMNET/DIS”, Computer Science Department, Naval Postgraduate School, October 24th, 1994.

[MARC80] Marcus, Mitchell P., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA, 1980.

[MARK96] Markowitz, Judith A., *Using Speech Recognition*, Prentice Hall, Upper Saddle River, NJ, 1996.

[MOOR89] Moore, Robert, Pereira, Fernando, Murveit, Hy, “Integrating Speech and Natural Language Processing,” Proceedings: Speech and Natural Language Workshop, February 1989.

[NAS95] National Academy of Sciences, *Virtual Reality - Scientific and Technological Challenges*, National Academy Press, Washington, D.C., 1995.

[NOBL95] Nobles, Joseph Anthony, and Garrova, James Francis "Design and Implementation of Real-Time, Deployable Three Dimensional Shiphandling Training Simulator," Master's Thesis, Naval Postgraduate School, Monterey, California, June, 1995.

[NUAN95] Nuance Communications, *Nuance Speech Recognition System Developer’s Manual*, Corona Corporation, Menlo Park, CA, 1995.

[OBYR95] O'Byrne, James E. "Human Interaction within a Virtual Environment for Shipboard Training," Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1995.

[RUDN89] Rudnicky, Alexander I., “The Design of Voice Driven Interfaces,” Proceedings: Speech and Natural Language Workshop, February 1989.

[SAGE86] Sager, Naomi, “Sublanguage: Linguistic Phenomenon, Computational Tool,” In R. Grishman and R. Kittredge (eds.), *Analyzing Language in Restricted Domains*, Lawrence Erlbaum, Hillsdale, NJ, 1986.

[SAGB87] Sager, Naomi, et al, *Medical Language Processing: Computer Management of Narrative Data*, Addison-Wesley Publishing Co., Menlo Park, CA, 1987.

[SAVA95] Savage-Carmona, Jesus, ”A Hybrid System with Symbolic AI and Statistical Methods for Speech Recognition”, Ph.D Dissertation, University of Washington, 1995.

[SAVB95] Savage-Carmona, Jesus, Holden, Alistair, and Billinghurst, Mark, “A Hybrid System with Symbolic AI and Statistical Methods for Speech Recognition,” Human Interface Technology Center, University of Washington, 1996.

[SCHA75] Schank, Roger C., *Conceptual Information Processing*, Amsterdam, North-Holland, 1975.

[SENE92] Seneff, Stephanie, “TINA: A NaturalLanguage System for Spoken Language Applications”, *Association for Computational Linguistics*, Vol 18, Num 1, 1992.

[SHAH96] Shahshahani, Ben, E-mail message from the Nuance Support Team, September 3, 1996.

[SGIA94] Silicon Graphics Inc. Document Number 007-1680-020, *IRIS Performer Programming Guide*, J Hartman and P. Creek, 1994.

[SGIB94] Silicon Graphics Inc. Document Number 007-1681-020, *IRIS Performer Reference Pages*, S Fischler, J. Helman, M. Jones, J. Rohlf, A. Schaffer and C. Tanner, 1994.

[SGIC94] Silicon Graphics Inc., Document Number 007-2590-001, *Developer Magics's RapidApp User Guide, 1994*.

[SGID95] Silicon Graphics Inc. Document Number 007-2282-001, *Speech Manager User's Guide*, March 1995.

[SGIE95] Silicon Graphics Inc., IRIS ViewKit Programmer's Guide, 1994.

[SHNE92] Shneiderman, Ben., *Designing the User Interface*, Addison-Wesley Publishing Co., Menlo Park, CA 1992.

[STEW96] Stewart, Bryan Christopher "Mounting Human Entities to Control and Interact with Networked Ship Entities in a Virtual Environment," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1996.

[STOR95] Storms, Russell L., *NPSNET-3D Sound Server: An Effective Use of the Auditory Channel,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[SUDK88] Sudkamp, Thomas A., *Languages and Machines - An Introduction to the Theory of Computer Science*, Addison-Wesley Publishing Company, Inc., Ney York, 1988.

[VEEC95] Veech, Thomas, Lecture Notes given at Stanford University, September 14, 1995.

[WALD95] Waldrop, Marianne, Pratt, Shirley M., Pratt, David R., and McGhee, Robert, Falby, John S., and Zyda, Michael J. "Real-time Upper Body Articulation of Humans in a Networked Interactive Virtual Environment," *The First Workshop on Simulation and Interaction in Virtual Environments (SIVE95)*, University of Iowa, Iowa City, IA, 13-15 July, 1995 pp. 210-214.

[WAUA90] Wauchope, Kenneth, "A Tandem Semantic Interpreter for Incremental Parse Selection," NRL Repart 9288, Naval Research Laboratory, July 2, 1990.

[WAUB94] Wauchope, Kenneth, “Eucalyptus: Integrating Natural Language Input with a Graphical User Interface,” NRL/FR/5510-94-9711, Naval Research Laboratory, February 25, 1994.

[WAUC96] Wauchope, Kenneth, “NAUTILUS Overview,” Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, January 30, 1996.

[YOUN92] Young, Douglas A., *Object-Oriented Programming with C++ and OSF/Motif*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[ZYDA93] Zyda, M., Pratt, D., Falby, J., Barham, P., and Kelleher, K., “NPSNET and the Naval Postgraduate School Graphics and Video Laboratory,” *Presence*, Vol. 2, No. 3, 1993.

[ZYDB95] Zyda, Michael J., Pratt, David R., Pratt, Shirley M., Barham, Paul T., Falby, John S. "NPSNET-HUMAN: Inserting the Human into the Networked Synthetic Environment," *Proceedings of the 13th DIS Workshop*, 18-22 September, 1995, Orlando, Florida.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

5. Director, Marine Corps Research Center.....2
   MCCDC, Code: C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

6. Director, Studies and Analysis Division.....2
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

7. Dr. Michael J. Zyda, Professor 2
   Computer Science Department Code CS/ZK
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Major Nelson D. Ludlow, Assistant Professor 2
   Computer Science Department Code CS/LD
   Naval Postgraduate School
   Monterey, CA 93943-5000

9. John S. Falby, Lecturer 2
   Computer Science Department Code CS/FJ
   Naval Postgraduate School
   Monterey, CA 93943-5000

10. Paul Barham.....1
    Computer Science Department Code CS/FJ
    Naval Postgraduate School
    Monterey, CA 93943-5000

11. Capt Edward M. DeVilliers 1
    489 Booth Hill Rd.
    Trumbull, CT 06611